

普通高等教育“十三五”规划教材
应用型特色规划教材

C 语言程序设计案例教程

肖利群 石 彬 主编

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

C 语言是一门入门语言,是学习其他计算机语言的基础。本书共 10 章,内容包括:初识 C 语言,C 语言案例概述,基本数据类型、运算符和表达式,数据的输入/输出,结构化程序设计,数组,函数,自定义数据类型,指针,文件。本书内容的讲解注重理论联系实际,以教师工资管理系统案例贯穿 C 语言各知识点的讲解,并在附录中给出教师工资管理系统的完整代码。本书包含精选的习题,学生通过练习不仅能掌握 C 语言相关理论知识,还能进一步训练编程能力。

本书可作为大学本科、高职高专学生“C 语言程序设计”课程的教学用书,同时也可作为参加计算机等级考试的人员和自学者的参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

C 语言程序设计案例教程 / 肖利群, 石彬主编. —北京: 电子工业出版社, 2018.9

ISBN 978-7-121-34746-7

I. ①C… II. ①肖… ②石… III. ①C 语言—程序设计—高等学校—教材 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2018)第 159485 号

策划编辑: 戴晨辰

责任编辑: 戴晨辰

印 刷:

装 订:

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 787×1092 1/16 印张: 19.25 字数: 493 千字

版 次: 2018 年 9 月第 1 版

印 次: 2018 年 9 月第 1 次印刷

定 价: 49.80 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话: (010)88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: dcc@phei.com.cn。

前言

Preface

C 语言是一门通用的计算机编程语言，它功能强大、使用灵活、应用广泛、目标程序效率高、可移植性好，适用于编写系统软件，深受编程人员欢迎。

作者自 2014 年起进行教学改革，变“单元制”教学为“理实一体”教学，更加注重知识点的学习及学生实践能力的培养。因此，通过多年的教学，作者积累了大量的“案例+项目”实训经验，特别是在“C 语言程序设计”课程中积累了较多的优秀案例、教学成果。

本书根据教学改革的变化进一步调整、完善了 C 语言的知识点和体系，以便更好地适应项目教学的授课要求。本书的编写目标是：集理论教学和实训教学于一体，针对计算机学科交叉、实践性强等特点，改变现有专业教材将各类知识点拆分、独立讲解的现状，编写理论讲解与实训案例相结合的教材，供计算机科学与技术、软件工程、网络工程、物联网、数字媒体等专业的学生使用。本书主要有以下特点。

(1) 涉及理论、概念等内容讲解时，注重穿插学习方法的介绍，结合应用型本科学生的特点，关注内容的实用性和综合性。删减以往同类图书中较刻板的理论知识点，将更多的学时和内容重点放在程序设计方法、设计技能及设计过程的阐述上。

(2) 突出“理实一体”的教学方法和过程展示，以教师工资管理系统案例贯穿 C 语言各知识点的讲解。将 C 语言的基本语法与知识点内嵌在案例中，并站在学生的角度，由浅入深、由表及里地罗列与讲解 C 语言相关知识，使学生循序渐进地学习 C 语言。

(3) 语言通俗易懂，难理解之处都配有图示。案例配有完整可用的程序代码，以便帮助学生在学知识点的同时，逐步提高编程能力。

(4) 在章节编排上力求循序渐进，在语言描述上力求准确、易懂，在案例实现上力求实用。

(5) 包含配套视频、源代码、教学课件等资源，读者可登录华信教育资源网（www.hxedu.com.cn）注册后免费下载。为帮助学生更好地学习相关知识点，本书还提供在线答疑（可从华信教育资源网获取答疑地址）。

本书由肖利群、石彬主编，第 3、4、5、6 章由肖利群编写，第 1、2、7、8、9、10 章由石彬编写，还要感谢李志、李爱华、谢治军、杨开林、郑丽娟、张德发、王邦千等老师对本书付出的辛勤劳动。本书在编写过程中，得到了四川工商学院领导的大力支持，同时得到了许多同行、专家的指导与帮助，在此对他们表示衷心的感谢。

本书难免存在不足之处，欢迎各位专家和广大读者提出宝贵意见。

作 者

2018 年 7 月于四川工商学院

目录 Contents

第 1 章 初识 C 语言	1
1.1 C 语言的发展历史和特点	1
1.1.1 C 语言的起源与发展	1
1.1.2 C 语言的特点	2
1.2 C 语言程序的基本结构	4
1.2.1 第一个 C 语言程序: Hello world!	4
1.2.2 基本结构	4
1.3 C 语言的集成开发环境	5
1.3.1 主流开发工具介绍	5
1.3.2 VC++ 6.0 环境介绍	6
1.3.3 C 语言程序的编译运行	8
1.4 本章小结	12
1.5 习题	12
第 2 章 C 语言案例概述	14
2.1 案例功能描述	14
2.1.1 输入记录模块	14
2.1.2 查询记录模块	15
2.1.3 更新记录模块	15
2.1.4 输出记录模块	15
2.2 案例总体设计	15
2.2.1 功能模块设计	15
2.2.2 数据结构设计	18
2.2.3 函数功能描述	19
2.3 案例运行结果	20
2.4 本章小结	24
第 3 章 基本数据类型、运算符和表达式	25
3.1 C 语言的字符集和词汇	25

3.1.1	C 语言的字符集	25
3.1.2	C 语言的词汇	26
3.2	常量和变量	28
3.2.1	常量	28
3.2.2	变量	29
3.3	数据类型	30
3.3.1	整型数据	31
3.3.2	实型数据	33
3.3.3	字符型数据	35
3.4	运算符与表达式	39
3.4.1	算术运算符与算术表达式	40
3.4.2	自增、自减运算符与表达式	40
3.4.3	关系运算符与关系表达式	41
3.4.4	逻辑运算符与逻辑表达式	42
3.4.5	赋值运算符与赋值表达式	43
3.4.6	逗号运算符与逗号表达式	44
3.4.7	条件运算符与条件表达式	45
3.4.8	sizeof 运算符与 sizeof 表达式	46
3.5	数据类型的转换	47
3.6	本章小结	48
3.7	习题	49
第 4 章	数据的输入/输出	54
4.1	输入/输出概述	54
4.2	非格式化字符的输入/输出	54
4.3	格式化数据的输出	55
4.4	格式化数据的输入	57
4.5	本章小结	59
4.6	习题	59
第 5 章	结构化程序设计	66
5.1	算法	66
5.1.1	算法的概念	66
5.1.2	结构化程序设计的三种基本结构	67
5.1.3	流程图	67
5.2	if 分支语句	69
5.2.1	if 语句中的条件表示	70
5.2.2	if 语句的三种形式	71
5.2.3	复合语句在分支语句中的应用	75
5.2.4	if 语句的嵌套	76

5.2.5	条件运算符与条件表达式	78
5.3	switch 分支语句	79
5.3.1	switch 语句	79
5.3.2	分支结构程序举例	83
5.4	循环结构	86
5.4.1	while 语句	86
5.4.2	do···while 语句	91
5.4.3	for 语句	93
5.4.4	break 语句和 continue 语句	99
5.4.5	多重循环结构	101
5.4.6	循环结构程序举例	104
5.5	本章小结	110
5.6	习题	111
第 6 章	数组	138
6.1	一维数组	138
6.1.1	一维数组的定义和初始化	138
6.1.2	一维数组的引用	142
6.1.3	一维数组程序举例	144
6.2	二维数组	150
6.2.1	二维数组的定义和初始化	150
6.2.2	二维数组的引用	153
6.2.3	二维数组程序举例	153
6.3	字符数组	156
6.3.1	字符数组的定义	156
6.3.2	字符数组的初始化	156
6.3.3	字符数组的赋值	157
6.4	字符串	158
6.4.1	字符串常量	158
6.4.2	利用字符串对字符数组初始化	158
6.4.3	字符数组与字符串的输入、输出	159
6.4.4	字符串处理函数	163
6.4.5	字符串程序举例	165
6.5	本章小结	168
6.6	习题	168
第 7 章	函数	179
7.1	函数的定义与调用	179
7.1.1	函数的分类	180
7.1.2	函数的定义	180

7.1.3	函数的调用	182
7.2	函数的参数传递	183
7.3	函数的调用方式	185
7.3.1	函数的嵌套调用	186
7.3.2	函数的递归调用	186
7.4	变量的作用域	189
7.4.1	变量的存储类型	189
7.4.2	全局变量与局部变量	190
7.5	编译预处理	192
7.5.1	文件包含	192
7.5.2	宏定义与替换	193
7.5.3	条件编译	195
7.6	本章小结	195
7.7	习题	196
第 8 章	自定义数据类型	204
8.1	结构体	204
8.1.1	结构体的定义	204
8.1.2	结构体变量的定义与初始化	205
8.1.3	结构体变量的引用	207
8.2	结构体数组	208
8.2.1	结构体数组的定义与初始化	208
8.2.2	结构体数组元素的引用	209
8.3	共用体	210
8.3.1	共用体的定义	210
8.3.2	共用体变量的定义与初始化	212
8.3.3	共用体变量的引用	212
8.4	本章小结	214
8.5	习题	215
第 9 章	指针	222
9.1	指针的概念	222
9.1.1	指针与指针变量	222
9.1.2	指针变量的引用	224
9.2	指针的运算	225
9.3	指针与数组	227
9.3.1	指向一维数组的指针	227
9.3.2	指向二维数组的指针	230
9.3.3	字符指针	232
9.4	指针与函数	233

9.4.1	指针和数组名作为函数参数	233
9.4.2	指针作为函数的返回值	240
9.5	链表	243
9.5.1	链表的概念	243
9.5.2	链表的基本操作	245
9.6	本章小结	250
9.7	习题	251
第 10 章	文件	261
10.1	文件概述	261
10.1.1	文件的概念	261
10.1.2	缓冲文件系统与非缓冲文件系统	262
10.1.3	文件指针	262
10.2	文件的打开与关闭	263
10.2.1	打开文件函数	263
10.2.2	关闭文件函数	264
10.3	文件的读与写	264
10.3.1	文件的写函数	265
10.3.2	文件的读函数	268
10.4	其他相关函数	273
10.5	本章小结	276
10.6	习题	276
附录 A	ASCII 码表	282
附录 B	运算符的优先级和结合性	283
附录 C	常用 ANSI C 标准库函数	284
附录 D	教师工资管理系统完整代码	290

第1章 初识 C 语言

C 语言是一门诞生较早的面向过程的高级程序设计语言。从诞生开始，由于其有着其他结构化程序设计语言所没有的优点，而深受广大编程人员的喜爱，并得到广泛使用。在 C 语言的基础上进行扩展，又衍生出 C++、C#等面向对象的程序设计语言。编程初学者多以 C 语言作为计算机编程学习的第一门语言。

本书将为读者揭开计算机编程的神秘面纱。本章作为本书的第 1 章，将针对 C 语言的发展历史和特点、C 语言程序的基本结构、C 语言程序的编译环境等内容进行详细讲解。

学习目标

- 了解 C 语言的发展历史和特点
- 掌握 C 语言程序的基本结构
- 掌握 C 语言程序的编写方式
- 熟悉开发工具 VC++ 6.0 的使用方法

1.1 C 语言的发展历史和特点

1.1.1 C 语言的起源与发展

在 C 语言诞生之前，计算机系统软件主要是用汇编语言编写的，但汇编语言编写的程序可读性和可移植性较差，已有的高级语言还没有对硬件直接访问的功能。因此，人们急需一种高级语言，使用它可以编写出可读性和可移植性强的程序，又能够直接操作硬件。这就为 C 语言的诞生提供了条件。

1963 年，剑桥大学在 ALGOL (Algorithmic Language) 的基础上开发出具有处理硬件能力的 CPL (Combined Programming Language)。

1967 年，剑桥大学的 Martin Richards 对 CPL 进行简化，于是产生 BCPL (Basic Combined Programming Language)。

1970 年，美国贝尔实验室的 Ken Thompson 以 BCPL 为基础，设计出简单且接近硬件的 B 语言(取 BCPL 的首字母)。

1972 年，美国贝尔实验室的 D. M. Ritchie 在 B 语言的基础上最终设计出了一种新的语言，他将 BCPL 的第 2 个字母作为这种语言的名称，这就是 C 语言。

随着计算机的发展和普及，C 语言受到越来越多的编程人员的喜爱，已经成为使用人数最多的结构化程序设计语言之一。随后，C 语言出现了多个版本，不同的版本之间存在一些不一致的地方。因此，美国国家标准学会(ANSI)为 C 语言制定了一套标准，即 ANSI C 标准。

1989 年, ANSI 通过的 C 语言标准称为 C89。

1990 年, ISO 组织批准了 ANSI C 成为国际标准, 于是 ISO C (又称 C90) 诞生了。ISO C (C90) 和 ANSI C (C89) 在技术上完全一样。

随后, ISO 在 1994 年、1996 年分别出版了 C90 的技术勘误文档, 更正了一些印刷错误, 并在 1995 年通过了 C90 的技术补充, 对 C90 进行了微小的扩充, 经过扩充后的 ISO C 称为 C95。

1999 年, ANSI 和 ISO 又通过了最新版本的 C 语言标准和技术勘误文档, 该标准称为 C99。

1.1.2 C 语言的特点

C 语言具有如下特点。

(1) 简洁、紧凑、灵活

C 语言的核心内容很少, 只有 32 个关键字, 9 种控制语句; 程序书写格式自由, 压缩了一切不必要的成分。

(2) 表达方式简练、实用

C 语言有一套强有力的运算符, 共 44 个, 可以构造出多种形式的表达式。使用一个表达式就可以实现其他语言需多条语句才能实现的功能。

(3) 具有丰富的数据类型

数据类型越多, 数据的表达能力就越强。C 语言具有现代语言的各种数据类型, 如字符型、整型、实型、数组、指针、结构体和共用体等。可以实现如链表、堆栈、队列、树等各种复杂的数据结构。其中指针的使用可使参数的传递更加简单、迅速, 节省内存。

(4) 具有低级语言的特点

C 语言具有与汇编语言相近的功能和描述方法, 如地址运算、二进制数位运算等, 对硬件端口等资源直接操作, 可充分使用计算机资源。C 语言既具有高级语言便于学习和掌握的特点, 又具有机器语言或汇编语言对硬件的操作能力。因此, C 语言既可以作为系统描述语言, 又可以作为通用的程序设计语言。

(5) 是一种结构化语言, 适合大型程序的模块化设计

C 语言提供编写结构化程序的基本控制语句, 如 `if...else` 语句、`switch` 语句、`while` 语句、`do...while` 语句等。C 语言程序是函数的集合, 函数是构成 C 语言程序的基本单位, 每个函数具有独立的功能, 函数之间通过参数传递数据。除了用户编写的函数, 不同的编译系统、操作系统还提供大量的库函数供用户使用, 如输入/输出函数、数学函数、字符串处理函数等, 灵活使用库函数可以简化程序设计。

(6) 各种版本的编译系统都提供预处理命令和预处理程序

预处理扩展了 C 语言的功能, 提高了程序的可移植性, 为大型程序的调试提供方便。

(7) 可移植性好

程序从一个环境不经改动或稍加改动就可移植到另一个完全不同的环境中运行。这是因为系统库函数和预处理程序将可能出现的与机器有关的因素与源程序隔离开, 使在不同的 C 语言编译系统之间重新定义有关内容变得容易。

(8) 生成的目标代码质量高

用 C 语言编写得到的目标代码的运行效率仅比用汇编语言编写的低 10%到 20%，可充分发挥机器的效率。

(9) 语法限制少，程序设计自由度高

C 语言程序在运行时不做如数组下标越界和变量类型兼容性等检查，而是由程序员自己保证程序的正确性。C 语言几乎允许所有数据类型的转换，字符型和整型可以自由混合使用，所有类型均可作为逻辑型，还可自己定义新的类型，将某类型强制转换为指定的类型。实际上，这使得程序员有了更大的自主性，可编写出更加灵活、优质的程序。但这也给初学者的学习增加了一定的难度，只有在熟练掌握 C 语言程序设计后，才能体会到其灵活的特性。

通过上述介绍，我们已经了解了 C 语言的若干特点。C 语言虽然是一种优秀的计算机程序设计语言，但也存在以下缺点，了解这些缺点，才能在实际使用中扬长避短。

(1) C 语言程序的错误更隐蔽

C 语言的灵活性使得程序员在编写程序时更容易出错，且 C 语言的编译器不会检查类似的错误。C 语言与汇编语言类似，需要在程序运行时才能发现这些逻辑错误。还有一些错误需要程序员重视，例如，将比较运算符“==”写成赋值运算符“=”，在语法上没有错，但这种逻辑错误不易被发现，要找出来往往十分费时。

(2) C 语言程序有时难以理解

C 语言语法成分相对简单，是一种小型语言。但是，其数据类型多，运算符丰富且结合性多样，使得理解起来有一定的难度。有关运算符和结合性，人们最常说的一句话是“先乘除，后加减，同级运算从左到右”，但是，C 语言的使用远比这要复杂。发明 C 语言时，为了减少字符输入，程序设计常比较简洁，这也使得 C 语言程序有时难以理解。

(3) C 语言程序有时难以修改

考虑到程序规模的大型化或巨型化，现代编程语言通常会提供“类”、“包”之类的语言特性，这样的特性可以将程序分解成更加易于管理的模块。然而，C 语言缺少这样的特性，维护大型程序显得比较困难。

早期的计算机语言有 BASIC 语言、Fortran 语言、ALGOL、COBOL 和 Pascal 语言等，如今除非是在既有的软件系统使用这些语言，其他时候很少使用。

现在的软件开发常使用 C++语言和 Java 语言，在 Web 应用软件开发时则会使用 JSP 语言和 PHP 语言等。随着面向对象技术的广泛普及，Java 语言受到更多人的青睐，这是由于 Java 语言具有编程效率高，可降低软件开发成本，不需要考虑存储的分配与回收等特点，编写出来的程序更具有健壮性，但也需在一定程度上付出运行效率的代价。C++语言介于 C 语言和 Java 语言之间，也是面向对象的计算机语言，具有编程效率高和运行速度快的特点。

C 语言是一种过程性的语言，职业程序员或软件开发人员应该认真学习该语言。这是因为，C 语言可以代替机器语言或汇编语言编写运行速度快的程序；对于单片机应用、嵌入式系统和通信软件等是不可替代的；C 语言的指针与计算机硬件的地址类似，是了解计算本质的钥匙；通过 C 语言的存储分配函数，可以深入了解计算机存储分配的原理。

1.2 C 语言程序的基本结构

1.2.1 第一个 C 语言程序：Hello world!

学习一门程序设计语言往往都是从从在屏幕上输出简单的信息开始的，一个输出简单信息的 C 语言程序如案例 1-1 所示。

【案例 1-1】 编写程序，在屏幕上输出 “Hello world!”。

```
#include<stdio.h>
void main()
{
    printf("Hello world!\n");
}
```

程序运行后的结果如图 1-1 所示。

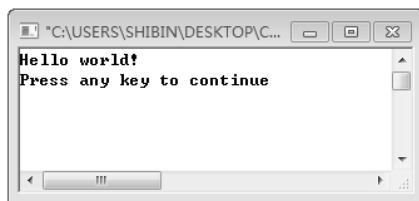


图 1-1 程序运行后的结果

1.2.2 基本结构

C 语言程序由一个或多个函数组成，每个函数是完成一定功能的一段 C 语言语句，如案例 1-1 中的 `main()` 函数。因此，构成 C 语言程序的基本单位是函数，在构成 C 语言程序的所有函数中必须有且只有一个 `main()` 函数，即主函数。也就是说，一个 C 语言程序必须由至少一个函数组成，这个函数就是主函数。程序运行总是从主函数开始，在主函数结束。

在 C 语言程序中，除主函数外，其他函数的函数名由程序员确定，称为自定义函数。每个函数由函数类型、函数名和函数名后圆括号内的参数组成函数首部，其后在花括号中由若干条功能语句构成函数体。在最简单的情况下，函数的格式如下：

```
函数类型 函数名(参数)
{
    函数体
}
```

其中，圆括号内的参数既可以有一个也可以有多个，还可以没有，但没有参数时，圆括号不能省略；函数体由若干条 C 语句构成，每条 C 语句以分号表示结束，而不以换行表示一条语句结束，多数程序员都习惯在一条语句结束的地方换行，以增强程序的可读性。

自定义函数不能像主函数那样独立运行，它只能由主函数或其他函数调用运行。所谓调用，就是函数暂时中断本函数的执行，转去执行被调用函数的功能语句。当被调用函数执行完或遇到 `return` 语句时，必须返回原先中断执行的函数，继续执行该函数后面的语句。在这个过程中会发生原函数调用被调函数和被调函数返回原函数的情况，可见，函数之间就是互相调用和返回的关系，如案例 1-2 所示。

【案例 1-2】 函数之间的调用与返回。

```
#include<stdio.h>
int add(int x,int y)
{
    return x+y;
}
int main()
{
    int a=50,b=40;
    int c;
    c=add(a,b); /* 调用 add()函数*/
    printf("a+b=%d\n",c);
    return 0;
}
```

在案例 1-2 中定义了一个两数相加的 `add()` 函数，其参数是 `x` 和 `y`，称为形式参数。当主函数执行到“`c=add(a,b);`”时，中断主函数的执行，去执行 `add()` 函数，`add()` 函数中实现 `x` 与 `y` 相加，并由 `return` 将两个数相加的结果带回到调用它的函数中，接着再执行主函数中的语句，程序执行后的结果是输出“`a+b=90`”。

总之，C 语言程序是由一个或多个函数组成的，其中必须有且仅有一个名为 `main()` 的主函数；程序的执行从主函数开始，在主函数结束，其他函数通过调用来执行；非主函数之间可以相互调用，但不能调用主函数；函数体是一段完成某个功能的 C 语句，每条语句由一个分号作为结束标志。



1.3 C 语言的集成开发环境

1.3.1 主流开发工具介绍

C 语言是一门计算机高级语言，用 C 语言编写的程序称为“源程序”，但计算机不能直接执行，因为计算机只能识别和执行由 0 和 1 组成的二进制的指令。为了使计算机能够执行以 `.C` 为扩展名的源程序，必须先编译源程序，将源程序翻译成以 `.obj` 为扩展名的二进制形式的目标程序，然后将目标程序与系统的函数库和其他目标程序链接起来，形成以 `.exe` 为扩展名的可执行程序。编译和链接的过程需要专用的编译工具，主要的编译工具有如下几种。

1. VC++ 6.0 工具

Microsoft Visual C++ 6.0，简称 VC++ 6.0，是微软于 1998 年推出的一款 C/C++ 编译器。其集成了 MFC 6.0，包含标准版 (Standard Edition)、专业版 (Professional Edition) 与企业版 (Enterprise Edition)。发行至今一直被广泛用于大大小小的项目开发。它不但具有程序框架自动生成、类管理灵活方便、代码编写和界面设计集成交互操作、可开发多种程序等优点，通过设置还可生成程序框架支持数据库接口、OLE 2.0、WinSock 网络。

2. Dev-C++工具

Dev-C++是一个 Windows 环境下的适合于初学者使用的轻量级 C/C++集成开发环境 (IDE)，它是一款自由软件，遵守 GPL 许可协议分发源代码。

Dev-C++使用 MingW64/TDM-GCC 编译器，遵循 C++ 11 标准，同时兼容 C++ 98 标准。开发环境包括多页面窗口、工程编辑器及调试器等，在工程编辑器中集合了编辑器、编译器、链接程序和执行程序，提供高亮度语法显示，以减少编辑错误，还有完善的调试功能，适合初学者与编程高手的不同需求，是学习 C 或 C++的首选开发工具。

3. Code::Blocks 工具

Code::Blocks 是一个开放源代码的全功能跨平台 C/C++集成开发环境，它最大的特点是支持通过插件的方式对 IDE 自身功能进行扩展，这使得 Code::Blocks 具有很强的灵活性，方便程序员使用。

Code::Blocks 本身并不包含编译器和调试器，它仅提供了一些基本的工具，用来帮助程序员从命令行中解放出来，使程序员享受更友好的代码编辑界面。不过，后期 Code::Blocks 的发行版本以插件的形式提供了编译和调试功能。

4. Turbo C 2.0 工具

Turbo C 2.0 不仅是一个快捷、高效的编译程序，同时还有一个易学、易用的集成开发环境。使用 Turbo C 2.0 无须独立编辑、编译和链接程序，就能建立并运行 C 语言程序。因为这些功能都组合在 Turbo C 2.0 的集成开发环境内，并且可以通过一个简单的主屏幕使用这些功能。但由于 Turbo C 2.0 不支持鼠标操作，对新的 Windows 系统兼容性不是很好，现在已经很少使用。

1.3.2 VC++ 6.0 环境介绍

全国计算机等级考试的考试大纲要求 C 语言的编辑环境是 VC++ 6.0，计算机专业的多数初学者也把 VC++ 6.0 作为编辑环境，这是由于 VC++ 6.0 在 Windows 下运行友好，学生更易于上手操作。本书选取 VC++ 6.0 作为 C 语言程序的开发工具。

1. VC++ 6.0 的安装

VC++ 6.0 是微软于 1998 年推出的一款 C/C++编译器，如今主流的硬件环境都能满足安装要求，对于软件系统，Windows 系统也能很好地支持。VC++ 6.0 在 Windows 7 系统中的安装步骤如下。

双击安装光盘中的 Setup.exe，启动安装向导，如图 1-2 所示。

单击“下一步”按钮，进入许可协议界面，选中“接受协议”后继续单击“下一步”按钮，进入“产品号和用户 ID”界面，如图 1-3 所示。输入相关信息后单击“下一步”按钮，在打开界面的“自定义-服务器安装程序选项”中选择“安装 Visual C++ 6.0 中文企业版(I)”，再单击“下一步”按钮，进入“选择公用安装文件夹”界面，如图 1-4 所示。单击“浏览”按钮设置安装路径后，单击“下一步”按钮，打开如图 1-5 所示的“Visual C++ 6.0 Enterprise 安装程序”对话框，选择“Typical”安装方式后开始复制相关文件进行安装。



图 1-2 安装向导



图 1-3 “产品号 and 用户 ID”界面

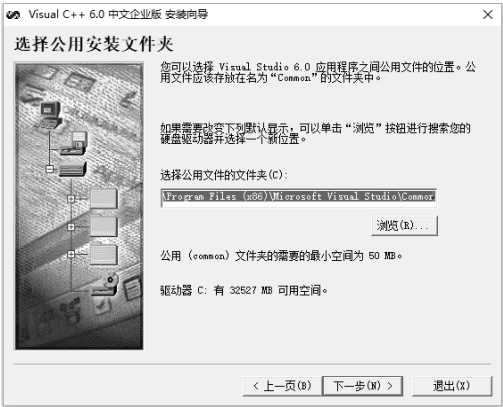


图 1-4 “选择公用安装文件夹”界面

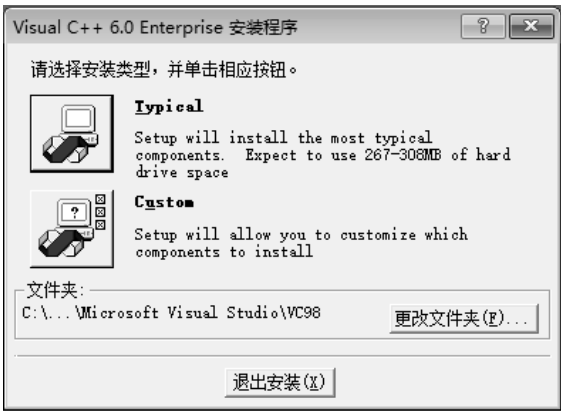


图 1-5 “Visual C++ 6.0 Enterprise 安装程序”对话框

安装完成后在系统的开始菜单中展开“Microsoft Visual C++ 6.0”，单击其中的“Microsoft Visual C++ 6.0”项就可以启动 VC++ 6.0，其主界面如图 1-6 所示。



图 1-6 VC++ 6.0 主界面

2. VC++ 6.0 的环境介绍

在图 1-6 所示的主界面中,主要包括菜单栏区域、工具栏区域、文件窗口区域、工作空间区域、输出窗口区域和状态栏区域,其中输出窗口区域默认状态是不打开的,在编译程序时会自动打开,也可以在“查看”菜单项中选中“输出”打开。

“菜单栏区域”包括文件、编辑、查看、插入、工程、组建、工具、窗口和帮助菜单项。“文件”菜单项主要用于与文件及工程相关的操作,可以新建工程或相应的源程序文件。“编辑”菜单项用于对 C 语言程序的各种编辑操作,主要包括文本的剪切、复制、粘贴、查找和替换等。“组建”菜单项主要用于 C 语言程序的编译和组建操作,可以将 C 文件编译组建成可以执行的程序。

“工具栏区域”默认打开标准工具栏、编译工具栏和向导工具栏,可实现一些与菜单栏区域相同的操作,如文件的新建、保存和编辑操作,程序的编译与组建操作等。

“文件窗口区域”主要用于输入 C 语言程序。当前没有打开任何文件,故不能输入程序。

“工作空间区域”主要用于显示当前工程的文件及函数构成。当前没有打开任何工程或项目,故显示为灰色。

“输出窗口区域”主要用于显示编译和组建程序的结果。在编译或组建时,正确或错误信息都会显示在输出窗口区域中。

“状态栏区域”主要用于显示源程序的编辑状态,包括源程序的编辑位置,所在行和列,以及覆盖或插入状态。

1.3.3 C 语言程序的编译运行

1. 创建工程项目

用 Visual C++ 6.0 系统建立 C 语言应用程序,首先要创建一个工程项目(project),用来存放 C 语言程序的所有信息。创建一个工程项目的操作步骤如下。

① 进入 Visual C++ 6.0 环境后,选择主菜单“文件”中的“新建”选项,在弹出的对话框中单击上方的“工程”选项卡,选择“Win32 Console Application”工程类型,在“工程名称”栏中填写工程名,如 first,在“位置”栏中填写工程路径(目录),如 D:\MYPROJECT\first,如图 1-7 所示,然后单击“确定”按钮。

② 随后将弹出“Win32 Console Application”对话框,选择“一个空项目”项,单击“完成”按钮。弹出“新建工程信息”对话框,单击“确定”按钮完成工程创建。创建的工作区文件为 first.dsw 和工程。

2. 新建 C 语言程序文件

选择主菜单“工程”中的“添加工程”命令中的“新建”选项,为工程添加新的 C 语言程序文件。打开如图 1-8 所示的“新建”对话框后,单击“文件”选项卡,选择“C++ Source File”项,在“文件名”栏填写新添加的源文件名,如 first.c,“位置”栏指定文件路径,单击“确定”按钮完成 C 语言程序的系统新建操作。注意:填写的源文件名一定要加上扩展名“.c”,否则系统会为文件添加默认的 C++源文件扩展名“.cpp”。此时文件窗口区域会自动打开 first.c 源程序,可对该程序进行编辑。




图 1-7 新建工程项目




图 1-8 添加新的 C 语言程序文件

3. 编译、链接和运行

(1) 编译

选择主菜单“组建”中的“编译”命令，或单击工具栏区域的“”图标，系统只编译当前文件而不调用链接器或其他工具。输出窗口将显示编译过程中检查出的错误或警告信息，在错误或警告信息处单击鼠标右键或双击鼠标左键，可以使输入焦点跳转到引起错误的源代码处，以便修改。如图 1-9 所示，输出窗口区域显示“...warning C4700: local variable 'c2' used without having been initialized”，提示本地变量没有初始化。

(2) 链接


选择“编译”中的“构建”命令，或单击工具栏区域的“”图标，对最后修改过的源文件进行编译和链接。

选择“编译”中的“重建全部”命令，允许用户编译所有的源文件，而不管它们何时曾被修改过。

选择“编译”中的“批构建”命令，能单步重新建立多个工程文件，并允许用户指定要建立的项目类型。

程序构建完成后生成的目标文件(.obj)、可执行文件(.exe)存放在当前工程项目所在文件夹的 Debug 子文件夹中。

(3) 运行

选择“编译”中的“执行”命令，或单击工具栏区域的“”图标，执行程序。将会出现一个新的用户窗口，按照程序输入要求正确输入数据后，程序即可正确执行，用户窗口显示运行的结果。对于比较简单的程序，可以直接选择该项命令，编译、链接和运行一次完成。

4. 调试程序

在编写较长的程序时，能够一次编写成功而没有任何错误绝非易事。对于程序中的错误，系统提供了易用且有效的调试手段。调试程序是一个程序员需要具备的最基本技能，不会调试就意味着即使学会了一门语言，也不能编写出任何好的程序。

(1) 调试程序环境

在菜单区空白处单击鼠标右键，在弹出的菜单中选择“调试”命令。激活“调试”工具栏，选择需要的调试命令，系统将会进入调试程序界面。同时会提供多种窗口监视程序运行，通过单击“调试”工具栏中的按钮，可以打开/关闭这些窗口，如图 1-10 所示。

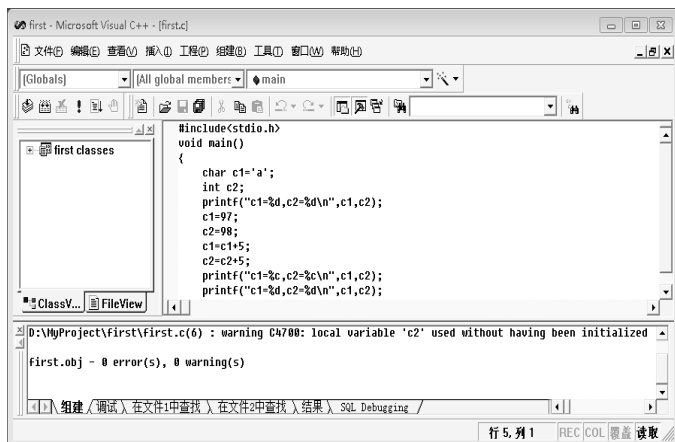


图 1-9 编译、链接和运行 C 语言程序

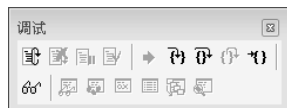


图 1-10 “调试”工具栏

10

单击“调试”工具栏中的 Watch 按钮，将弹出 Watch 窗口，系统支持查看程序运行到当前指令语句时变量、表达式和内存的值。所有这些观察都必须在断点中断的情况下进行。查看变量的值最简单，当断点到达时，把光标移动到这个变量上，停留一会儿就可以看到变量的值。还可以采用系统提供的 Watch 机制来查看变量和表达式的值。断点中断状态下，在变量上单击鼠标右键，选择 Quick Watch，弹出的对话框将显示这个变量的值。

单击“调试”工具栏中的 Variables 按钮，将弹出 Variables 窗口，显示所有当前执行上下文中可见的变量的值。特别是当前指令语句涉及的变量，将以红色显示。

指针指向的数组，Watch 窗口只能显示第 1 个元素的值，为了显示数组的后续内容，或者要显示一片内存的内容，可以使用 Memory 功能。单击“调试”工具栏中的 Memory 按钮，在弹出的对话框中输入地址，就可以显示该地址指向的内存的内容。

单击“调试”工具栏中的 Registers 按钮，将弹出 Registers 对话框，显示当前所有寄存器的值。

调用堆栈反映了当前断点处函数是被哪些函数、按照什么顺序调用的。单击“调试”工具栏中的 Call Stack 按钮，将弹出 Call Stack 对话框。在 Call Stack 对话框中将显示一个调用系列，最上面的是当前函数，向下依次是调用函数的上级函数。单击这些函数名可以跳转到对应的函数中去。

(2) 单步执行调试程序

系统提供多种单步执行调试程序的方法，可以通过单击“调试”工具栏中的按钮或使用快捷键的方式选择多种单步执行命令。


单步跟踪进入子函数(Step Into)：每按一次 F11 键，程序将执行一条无法再进行分解的程序行，如果涉及子函数，进入子函数内部；单步跟踪跳过子函数(Step Over)：每按一次 F10 键，程序执行一行；Watch 窗口可以显示变量名及其当前值，在单步执行过程中，可以在 Watch 窗口中加入要观察的变量，辅助监视，随时了解变量当前的情况，如果涉及子函数，不进入子函数内部；单步跟踪跳出子函数(Step Out)：按 Shift+F11 组合快捷键后，程序运行至当前函数的末尾，然后从当前子函数跳转到上一级主调函数；按 Ctrl+F10 组合快捷键后，程序将运行至当前光标处所在的语句。常用的调试命令如表 1-1 所示。

表 1-1 常用的调试命令

菜单命令	工具栏按钮	快捷键	说 明
Go		F5	继续运行，直到断点处中断
Step Over		F10	单步，如果涉及子函数，不进入子函数内部
Step Into		F11	单步，如果涉及子函数，进入子函数内部
Run to Cursor		Ctrl+F10	运行至当前光标处
Step Out		Shift+F11	运行至当前函数的末尾，跳转到上一级主调函数
Insert/Remove Breakpoint		F9	设置/取消断点
Stop Debugging		Shift+F5	结束程序调试，返回程序编辑环境

(3) 设置断点调试程序

为方便较大规模程序的跟踪，设置断点是最常用的技巧之一。断点是调试器设置的一个代码位置，当程序运行到断点时，程序中断执行，回到调试器。调试时，只有设置了断点并使程序回到调试器，才能对程序进行在线调试。

可以通过下述方法设置一个断点：首先把光标移动到需要设置断点的代码行上，然后按 F9 键或者单击“编译”工具栏中的“”按钮，断点处所在的程序行的左侧会出现一个红色圆点。

选择“编译”中的“开始调试”命令中的“GO”调试命令，或者直接按 F5 键，程序执行到第 1 个断点处时将暂停执行，该断点处所在的程序行的左侧红色圆点上将添加一个黄色箭头，此时，用户可方便地进行变量观察。继续执行该命令，程序运行到下一个相邻的断点，如图 1-11 所示。

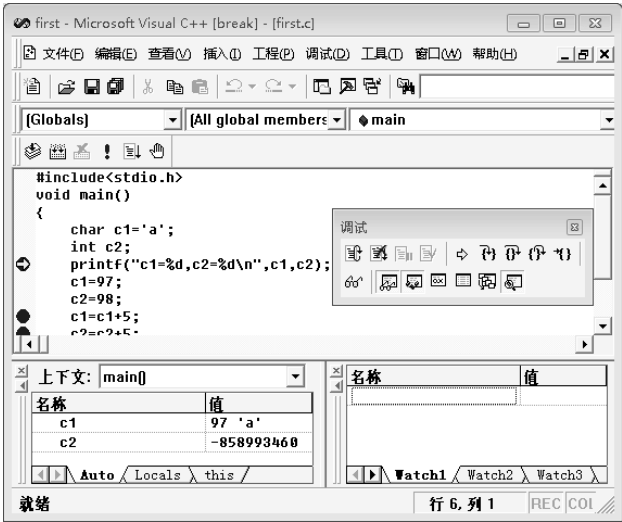




图 1-11 设置断点调试程序

要取消断点，只需在代码处再次按 F9 键，或单击“编译”工具栏中的“”按钮，也可以在打开 Breakpoints 对话框后，按照提示去掉断点。

选择“调试”中的“Stop Debugging”命令，或单击“调试”工具栏中的“”按钮，或按 Shift+F5 组合快捷键，可结束程序调试，返回程序编辑环境。

1.4 本章小结

通过本章内容的学习，我们初步了解了 C 语言的起源、发展及特点，掌握了 C 语言函数的基本结构，了解了主流的 C 语言集成开发环境，学习了 VC++ 6.0 集成开发环境的使用，最后通过创建 C 语言程序理解 VC++ 6.0 的编译与组建程序的过程，了解了 VC++ 6.0 调试程序的过程与方法。

1.5 习题

一、单选题

1. 以下不是 C 语言的特点的是()。
A. 语言简洁紧凑
B. 能够编写出功能复杂的程序
C. C 语言可以直接对硬件操作
D. C 语言移植性好
2. 一个 C 语言程序是由()。
A. 一个主程序和若干个子程序组成的
B. 一个或多个函数组成的
C. 若干个过程组成的
D. 若干个子程序组成的
3. C 语言程序的基本单位是()。
A. 程序行
B. 语句
C. 函数
D. 字符
4. 下列说法中错误的是()。
A. 每个语句必须独占一行，语句的最后可以是一个分号，也可以是一个回车换行符号
B. 每个函数都有一个函数头和一个函数体，主函数也不例外
C. 主函数只能调用用户函数或系统函数，用户函数可以相互调用
D. 程序是由若干个函数组成的，但是必须有且只有一个主函数
5. 以下说法中正确的是()。
A. C 语言程序总是从第一个定义的函数开始执行的
B. 在 C 语言程序中，要调用的函数必须在 `main()` 函数中定义
C. C 语言程序总是从 `main()` 函数开始执行的
D. C 语言程序中的 `main()` 函数必须放在程序的开始部分
6. C 编译程序是()。
A. C 语言程序的机器语言版本
B. 一组机器语言指令
C. 将 C 语言源程序编译成目标程序
D. 由制造厂家提供的一套应用软件
7. 以下关于 C 语言的描述正确的是()。
A. 在对一个 C 语言程序进行编译的过程中，可发现注释中的拼写错误
B. C 语言有输入/输出语句
C. 在 C 语言程序中，`main()` 函数必须位于程序的最前面
D. 在 C 语言程序中，函数不可以嵌套定义

8. 用 C 语言编写的代码程序()。

A. 可立即执行

B. 是一个源程序

C. 经过编译即可执行

D. 经过编译解释才能执行

二、填空题

1. C 语言程序一般由若干个函数构成，程序中应至少包含一个_____，其名称只能为_____。
2. C 语言程序中每条语句必须以_____结束。
3. C 语言程序的注释是以_____开头，以_____结束的，在 VC++ 6.0 编程环境中，可使用_____作为注释的起始标识，注释对程序的执行不起任何作用。
4. 最初编写的 C 语言程序称为_____，其扩展名为_____，编译后生成的文件为_____，其扩展名为_____，链接后生成的文件为_____，其扩展名为_____。

第2章 C语言案例概述

本书以教师工资管理系统案例贯穿全书，编写程序实现教师及其工资的管理。本章主要介绍教师工资管理系统的功能、总体设计、功能模块设计与实现。通过本章内容的学习使读者对信息管理系统的开发流程能有初步的了解，本程序中涉及的 C 语言知识会在后续章节进行具体讲解。读者在学习本章时可以主要关注程序中实现相应功能的流程，忽略 C 语言语法的细节，为开发高质量信息管理系统打下基础。

学习目标

- 了解案例功能模块的结构
- 了解案例功能模块的实现方法
- 了解教师工资信息的基本结构
- 了解案例功能函数的相关功能
- 了解案例的运行过程

2.1 案例功能描述

教师工资管理系统案例中教师工资由教师的基本工资和课时费构成，教师的职称不同，其所对应的基本工资和每节课的课时费也不同，教师基本工资和课时费如表 2-1 所示。教师每月有 40 学时的基本工作量，即如果教师的每月课时数小于 40 时，其工资计算公式为：教师工资=基本工资-扣款；如果教师每月课时数大于或等于 40 时，其工资计算公式为：教师工资=基本工资+(课时-40)×课时费-扣款。

表 2-1 教师基本工资和课时费

教师职称	基本工资	课时费
教授	4200	120
副教授	3800	100
讲师	3400	80
助教或无职称	3000	60

教师工资信息主要用数组来实现，其数组元素是结构体类型，整个系统由输入记录、查询记录、更新记录和输出记录几大功能模块组成，总体结构如图 2-1 所示。

2.1.1 输入记录模块

输入记录模块主要完成将数据存入数组中的工作。在教师工资管理系统中，记录可以从以文本形式存储的数据文件中读入，也可从键盘逐个输入记录。记录由教师有关基本信息字段构成。当从数据文件中读入记录时，它就是在以记录为单位存储的数据文件中，将记录逐条复制到结构体类型的数组元素中。

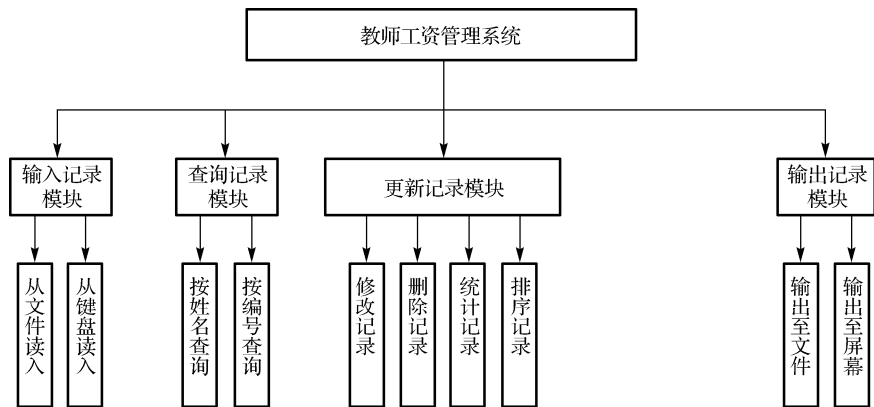


图 2-1 教师工资管理系统功能模块

2.1.2 查询记录模块

查询记录模块主要完成在数组中查找满足相关条件的记录。在教师工资管理系统中，用户可以按照教师姓名或教师编号在数组中进行查找。若找到该记录，则以表格形式显示出此记录的信息；否则，打印出未找到该记录的提示信息。

2.1.3 更新记录模块

更新记录模块主要完成记录的维护工作。在教师工资管理系统中，实现对教师工资信息的修改、删除、统计和排序操作。一般而言，系统在执行除统计以外的操作后，需要将修改的数据存入源数据文件，统计会随着记录的更新而更新。

2.1.4 输出记录模块

输出记录模块主要完成两个任务：第一，实现记录的存盘操作，即将数组各元素中存储的记录信息写入数据文件；第二，将数组中存储的记录信息以表格的形式在屏幕上显示出来。

2.2 案例总体设计

2.2.1 功能模块设计

1. 主控 main() 函数的执行流程

教师工资管理系统中 main() 函数的执行流程如图 2-2 所示。它先以可读/写的方式打开二进制类型的数据文件，此文件默认为“D:\demo\Salary.dat”，若该文件不存在，则新建此数据文件。当打开文件操作成功后，循环读取文件中的记录，循环一次读出一条记录，添加到数组的元素中，直到读取到数据文件的结尾，然后进入主循环，执行显示主菜单操作，并提示用户输入操作选择，根据用户选择的操作调用相应的功能函数，执行相应功能后，回到主菜

单或退出系统。值得一提的是，文本类型文件与二进制类型文件不同，文本类型文件可以使用 Windows 自带的记事本打开，并查看存储的文件内容，而二进制类型文件打开后无法识别其文件内容。

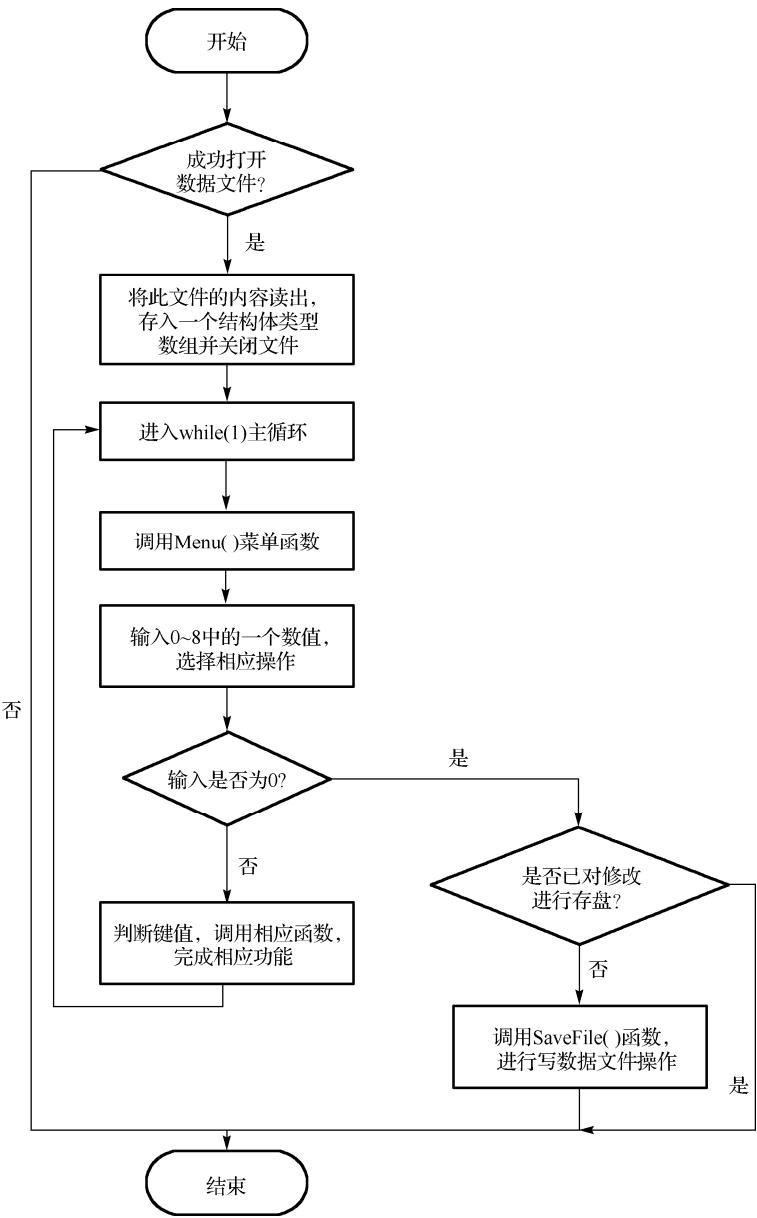


图 2-2 main() 函数的执行流程

在判断键值时，有效的输入为 0~8 之间的整数，其他输入都被视为错误按键。若输入 0（即变量 choice=0），则会继续判断记录进行更新操作后是否进行了存盘操作，若未存盘，则全局变量 saveflag=1，系统会提示用户是否需要进行数据存盘，用户输入 Y 或 y，系统进行存盘操作。最后，系统执行退出管理系统的操作。若输入 1，则调用 Add() 函数，执行增加记录操作。若输入 2，则调用 ShowRecord() 函数，执行将记录以表格形式输出至屏幕的操作。若输入 3，

则调用 `Modify()` 函数, 执行修改记录操作。若输入 4, 则调用 `Del()` 函数, 执行删除记录操作。若输入 5, 则调用 `Query()` 函数, 执行查询记录操作。若输入 6, 则调用 `Sort()` 函数, 执行按升序排序记录的操作。若输入 7, 则调用 `Total()` 函数, 执行统计记录操作。若输入 8, 则调用 `SaveFile()` 函数, 执行将记录存入磁盘中的数据文件的操作。若输入 0~8 以外的值, 则给出按键错误提示, 并重新回到选择界面。

2. 输入记录模块设计

输入记录模块主要实现将数据存入数组。当读出数据文件中的记录时, 调用 `Fread()` 文件读取函数, 执行一次从文件中读取一条教师工资记录存入某个数组元素中的操作, 并且这个操作在 `main()` 中调用执行, 即在教师工资管理系统进入显示菜单界面时, 该操作已经执行了。若该文件中没有数据, 则系统会提示数组为空, 没有任何记录可操作, 此时, 用户应输入 1, 调用 `Add()` 函数, 进行记录的输入, 完成在数组中添加元素的操作。

3. 查询记录模块设计

查询记录模块主要实现在数组中按教师姓名或教师编号查找满足条件的记录。在查询函数 `Query()` 中, 由于按教师姓名查询可能会出现同名的教师, 而按教师编号查询则不会出现重复的记录, 且在修改或删除记录时也需要按教师编号先查询到满足条件的记录后再进行修改或删除记录的操作。因此, 为了遵循模块化编程的原则, 将在数组中进行的某一个记录定位操作设计成一个单独的函数 `Locate()`, 若找到该记录, 则返回指向该记录的数组元素的下标; 否则, 返回 -1。按教师姓名查询是对数组元素的教师字段从第 1 个元素开始进行循环查找, 然后将满足条件的记录输出到屏幕上。

4. 更新记录模块设计

更新记录模块主要实现对记录的修改、删除、排序和统计操作。因为记录是以数组的结构形式存储的, 所以这些操作都在数组中完成。下面分别介绍这 4 个操作。

(1) 修改记录

修改记录操作需要对数组中目标元素的数据域中的值进行修改, 分两步完成。第一步, 输入要修改的教师编号, 输入后调用定位函数 `Locate()`, 在数组中逐个对教师编号字段的值进行比较, 直到找到该教师编号的记录; 第二步, 若找到该记录, 修改该记录各字段的值, 并将存盘标记 `saveflag` 置 1, 表示记录已修改, 但还未执行存盘操作。

(2) 删除记录

删除记录操作实现删除指定教师编号记录, 分两步完成。第一步, 输入要删除的教师编号, 输入后调用定位函数 `Locate()`, 在数组中逐个对教师编号字段的值进行比较, 直到找到该教师编号的记录, 返回指向该记录数组元素的下标; 第二步, 找到该记录后, 从该记录所在元素的后一个元素起, 依次将元素向前移一个位置, 有值的数组元素个数减 1。例如, 在删除数组元素 A2 后, 数组元素 A3 和 A4 向前移动了一个位置, 具体过程如图 2-3 所示。

(3) 排序记录

冒泡排序法属于内部排序法中的一种, 它是将相邻两个元素的排序字段值比较后, 依原则交换数组元素位置, 更新欲排序的数组元素, 以达到排序的目的。这里我们采用冒泡排序法来实现教师编号或教师工资字段数值从低到高排序(升序排序)。

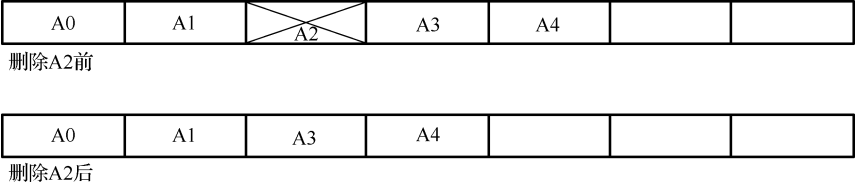


图 2-3 数组中删除记录示意图

冒泡排序的方法为：将相邻的两个数组元素的教师编号或教师工资字段的值进行比较，若左边的值大于右边的值，则将两个元素的值进行交换；若左边的值小于或等于右边的值，则两个值的位置不变。右边的值继续和下一个值做比较，重复此动作，直到比较到最后一个值为止，最终实现升序排序。冒泡排序法是最简单的排序法，但所需的排序时间比其他排序法长。

(4) 统计记录

统计所有记录的信息包括：统计各类职称教师的人数及占教师总数的比例、最高工资、最低工资和平均工资。从第 1 条记录开始逐个对教师的职称字段进行判断，若符合，则在相应职称人数的变量上加 1。对教师工资字段进行比较，将工资最大值和最小值分别存储在相应的变量中，并将教师的工资字段累加到工资总额的变量中。在统计完成后用记录职称人数的变量除以总的教师人数，得到所占比例，用所有教师的工资总额除以总的教师人数，得到全校教师的平均工资。

5. 输出记录模块设计

当把记录输出至文件时，调用 `Fwrite()` 函数，将数组元素中各字段的值逐个写入文件指针 `fp` 所指的文件；当把记录输出至屏幕时，调用 `ShowRecord()` 函数，将数组中存储的记录信息以表格的形式逐个在屏幕上显示出来。

2.2.2 数据结构设计

以下程序定义了结构体 `TeaSalary`，用于存放教师工资的基本信息：

```
struct TeaSalary
{
    int TeaNum;
    char TeaName[13];
    char TeaTitle[7];
    int BasePay;
    int ClassHours;
    int ClassHoursSubsidy;
    int Debit;
    int Salary;
}tea[100];
```

各字段值的含义如下。

`TeaNum`：保存教师的编号。

`TeaName[13]`：保存教师的姓名。

`TeaTitle[7]`：保存教师的职称。

BasePay: 保存教师的基本工资。

ClassHours: 保存教师的课时数。

ClassHoursSubsidy: 保存教师的每节课课时费。

Debit: 保存教师工资的扣款。

Salary: 保存教师的实发工资。

2.2.3 函数功能描述

(1) Add() 函数

函数原型:

```
int Add(int cnt)
```

用于向 tea 数组中增加教师工资记录。cnt 是已有记录数, 作为新增加元素的起始下标。

(2) CalSalary() 函数

函数原型:

```
int CalSalary(int basepay,int classhours,int subsidy,int debit)
```

用于计算教师的应发工资。basepay 是教师的基础工资, classhours 是教师的课时数, subsidy 是教师的每节课课时费, debit 是教师工资的扣款。

(3) Del() 函数

函数原型:

```
int Del(int cnt)
```

用于教师工资记录的删除操作。cnt 是已有记录数。

(4) ExitSystem() 函数

函数原型:

```
void ExitSystem(int cnt)
```

用于退出系统操作。cnt 是已有记录数, 退出时如果保存记录, 则作为保存的记录数。

(5) JudgeBasePay() 函数

函数原型:

```
int JudgeBasePay(char title[])
```

用于根据教师职称确定教师的基本工资。title 是教师的职称。

(6) JudgeSubsidy() 函数

函数原型:

```
int JudgeSubsidy(char title[])
```

用于根据教师职称确定教师的课时费。title 是教师的职称。

(7) Locate() 函数

函数原型:

```
int Locate(int num,int cnt)
```

用于按教师编号查询教师信息。num 存放要查询的教师编号, cnt 是查询教师记录数。

(8) Menu() 函数

函数原型:

```
void Menu()
```

用于显示系统主菜单。

(9) Modify() 函数

函数原型:

```
void Modify(int cnt)
```

用于修改教师工资信息。

(10) OpenFile() 函数

函数原型:

```
int OpenFile(int cnt)
```

用于打开数据文件的操作。cnt 用于记录教师工资的记录数。

(11) Query() 函数

函数原型:

```
void Query(int cnt)
```

用于按教师编号或教师姓名查询记录。

(12) SaveFile() 函数

函数原型:

```
void SaveFile(int cnt)
```

用于保存数组中的 cnt 条记录的操作。

(13) ShowRecord() 函数

函数原型:

```
void ShowRecord(int cnt)
```

用于显示数组中的 cnt 条记录的操作。

(14) Sort() 函数

函数原型:

```
void Sort(int cnt)
```

用于对数组中的 cnt 条记录排序的操作。

(15) Total() 函数

函数原型:

```
void Total(int cnt)
```

用于对数组中的 cnt 条记录汇总的操作。



2.3 案例运行结果

当用户刚进入教师工资管理系统时,主界面如图 2-4 所示。此时,系统已经将“D:\demo\

Salary.dat”文件打开，若文件不为空，则将数据从文件中逐条读出，并写入数组。用户可输入0~8之间的数值，调用相应功能进行操作。当输入0时，系统判断saveflag变量的值为1还是0，如果为1，说明信息数据已经发生变化，会要求用户选择保存还是不保存文件，然后再退出此管理系统。

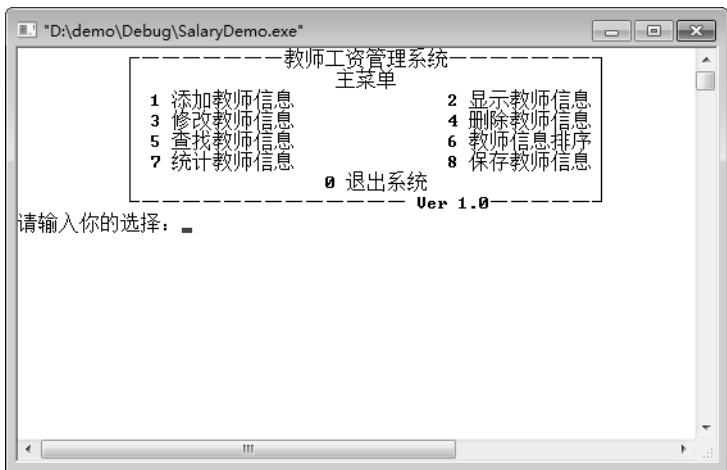


图 2-4 教师工资管理系统主界面

当用户输入1并按Enter键后，即可进入数据输入界面，其输入记录过程如图2-5所示。当输入完一条记录后会提示用户是否继续输入，如果输入1，继续输入记录；如果输入0，会结束输入过程，返回主菜单界面。

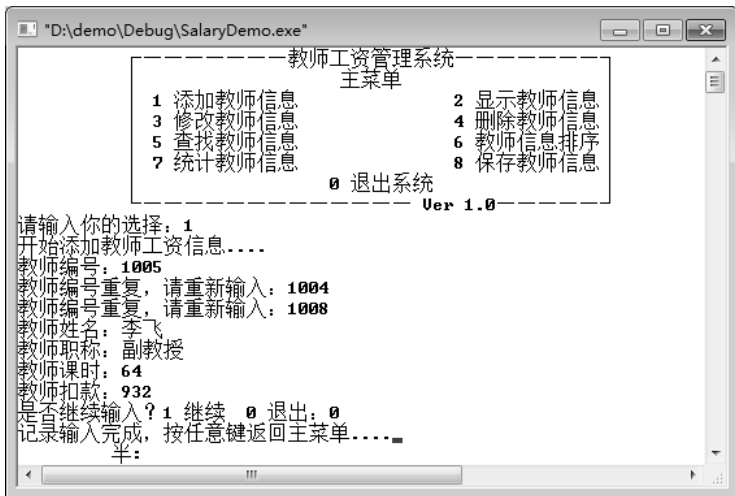


图 2-5 输入记录过程

当用户执行输入记录或已经从数据文件中读取了记录，则可输入2并按Enter键，查看当前数组中的记录情况，如图2-6所示，此时表中有8条记录。

当用户输入3并按Enter键后，即可进入记录修改界面。用户在查找界面输入要修改的教师编号，进行记录查找，如果没查找到，则提示“要修改的教师信息不存在！”，如果查找到要修改的记录，则在显示该记录后进入修改状态，重新输入记录的信息，如图2-7所示。

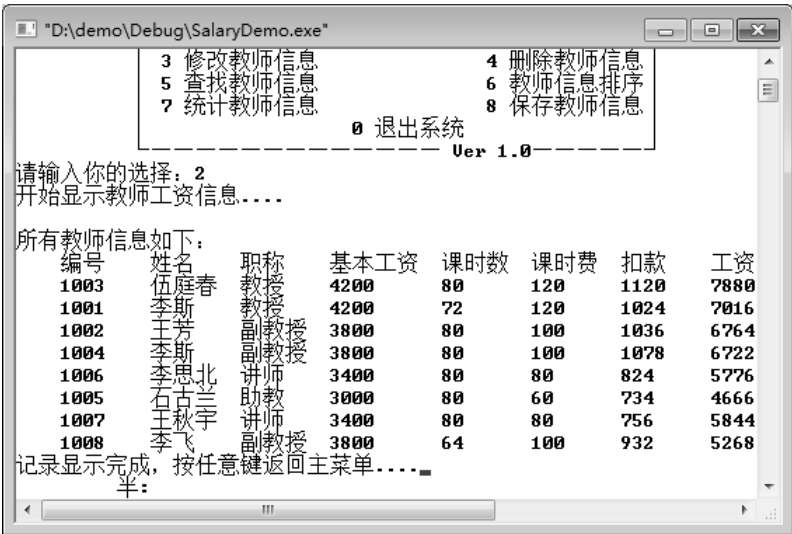


图 2-6 记录情况

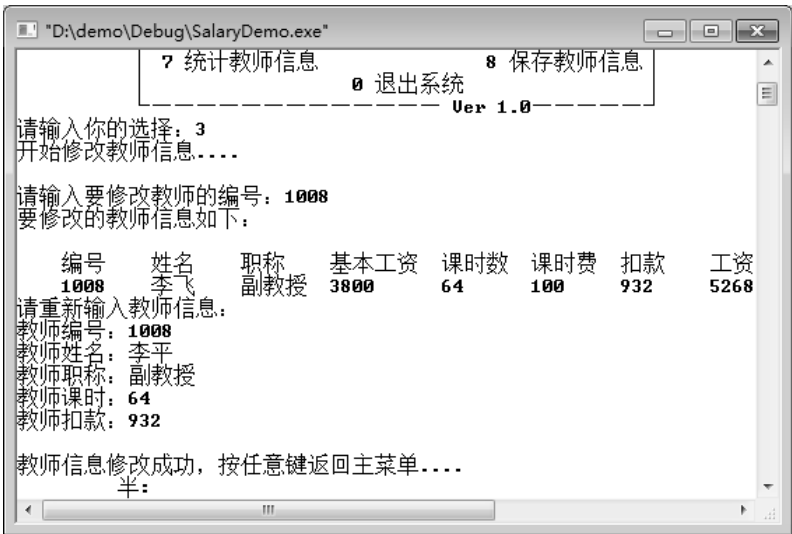


图 2-7 修改记录

当用户输入 4 并按 Enter 键后, 即可进入记录删除界面。用户在查找界面输入要删除的教师编号, 进行记录查找, 如果没查找到, 则提示“要删除的教师信息不存在!”, 如果查找到要修改的记录, 则在显示该记录后将该记录删除, 如图 2-8 所示。

当用户输入 5 并按 Enter 键后, 即可进入记录查询界面。在查询界面, 用户可以输入 1, 按教师编号查询, 也可以输入 2, 按教师姓名查询。在用户输入 1 或 2 并按 Enter 键后, 系统提示用户输入编号或姓名, 如果没查找到, 则提示“要查询的教师信息不存在!”, 如果查找到, 则显示所有记录。按姓名查询如图 2-9 所示。

当用户输入 6 并按 Enter 键后, 即可进入记录排序界面。用户可以选择按照教师编号或教师工资进行排序, 图 2-10 为按教师工资降序排序的结果。

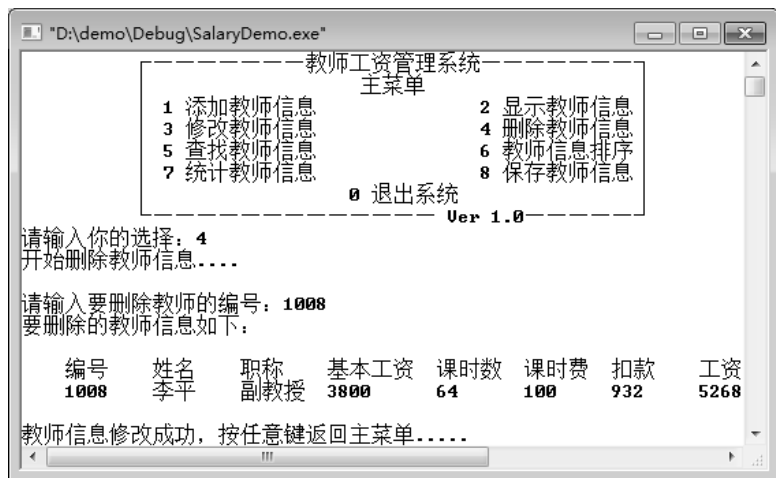


图 2-8 删除记录

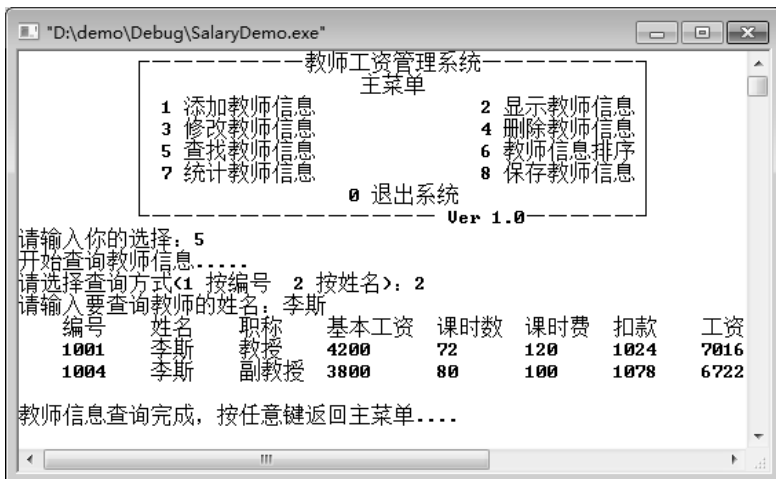


图 2-9 按姓名查询

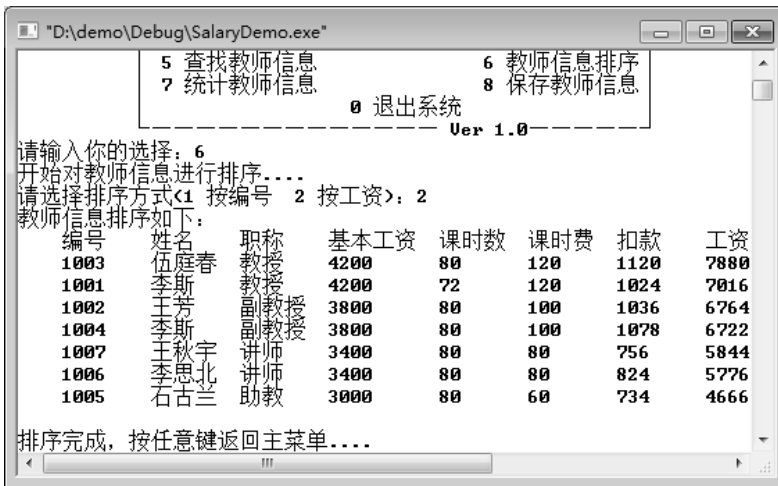


图 2-10 按教师工资降序排序

当用户输入 7 并按 Enter 键后，即可进入统计教师信息界面。可以统计出各职称的教师人数及所占比例、最高工资、最低工资的教师和全校教师平均工资。图 2-11 为统计教师信息的结果。



图 2-11 统计教师信息

当用户输入 8 并按 Enter 键后，系统将会把数组元素保存到 “D:\demo\Salary.dat” 中，并给出保存文件成功的提示，然后返回主菜单。

2.4 本章小结

本章介绍了教师工资管理系统的设计思路及其编程实现，重点介绍了各功能模块的设计原理，以及利用数组存储结构实现教师工资管理的过程。通过本章的案例概述使读者熟悉 C 语言下管理信息系统的开发过程，了解管理信息系统的开发原理，以及一些简单的关于功能实现的算法描述。

第3章 基本数据类型、运算符和表达式

本章主要介绍 C 语言的字符集、标识符、关键字、常量、变量、数据类型、运算符、表达式，以及数据类型的转换等。

学习目标

- 掌握 C 语言的字符集、标识符和关键字
- 掌握常量的表示方式
- 掌握变量的定义和使用方法
- 理解 C 语言的数据类型
- 掌握运算符及表达式的用法
- 理解运算符的优先级和结合性
- 掌握数据类型的转换规则，以及强制类型转换的方法

3.1 C 语言的字符集和词汇

3.1.1 C 语言的字符集

字符是组成语言最基本的元素，程序设计语言都有其各自的字符集。编写源程序必须使用指定语言字符集中提供的字符，而不能使用字符集以外的字符，否则编译系统会认为是非法字符。

C 语言字符集由字母、数字、空格、标点符号和特殊字符组成。在字符常量、字符串常量和注释中还可以使用汉字或其他可表示的图形符号等，主要介绍以下内容。

(1) 字母

大写字母 A~Z，小写字母 a~z。

(2) 数字

0~9，共 10 个。

(3) 空白符

空格符、制表符、换行符等统称为空白符。空白符只在字符常量和字符串常量中起作用。在其他地方出现时，只起间隔作用，编译程序将忽略它们。因此，在程序中使用空白符与否，对程序的编译不发生影响，但在程序中适当的地方使用空白符，可提高程序的清晰度，增强可读性。

(4) 运算符、标点符号，以及其他字符

+ - * / % = ! # ^ < > ; : & ~
| ? , . ' " \ () [] { } _ 空格

3.1.2 C 语言的词汇

在 C 语言中使用的词汇可分为 6 类：标识符、关键字、运算符、分隔符、常量、注释符。

1. 标识符(Identifier)

在程序中使用的变量名、函数名、数组名、类型名等统称为标识符。除库函数的函数名由系统定义外，其余都由用户自定义。

C 语言规定：

① 标识符只能是字母(A~Z, a~z, 区分大小写)、数字(0~9)、下画线(_)组成的字符串；

② 第一个字符必须是字母或下画线。

以下标识符是合法的：

saveFlag _teacher number_2 x5 A6b

以下标识符是非法的：

6p (不能以数字开头)

A&B (不能使用非法字符&)

c.3x (不能使用非法字符.)

sum-1 (不能使用非法字符-)

if (if 为关键字，不能用作标识符)

在使用标识符时还必须注意以下几点。

① 标准 C 不限制标识符的长度，但它受各种版本的 C 语言编译系统限制，同时也受到具体机器的限制。

② 在标识符中，大小写是有区别的，如 A 和 a 是两个不同的标识符。

③ 标识符虽然可由程序员按规则随意定义，但标识符是用于标识某个量的符号，并不是合法的标识符就是好的标识符，如定义一个教师变量，t1 显然没有 teacher 好。因此，命名时应尽量注意对应相应的意义，以便阅读理解，做到“见名知意”，可读性好。

2. 关键字(Keyword)

关键字是由 C 语言规定的具有特定意义的字符串，通常也称为保留字，所有的关键字都必须是小写英文字母。用户定义的标识符不能与关键字相同。

ANSI C 规定 C 语言共有 32 个关键字，如表 3-1 所示。

这些关键字的用法将在后续章节进行详细介绍。需要说明的是，不同的 C 编译系统对 C 标准的支持是有区别的，通常还会适当增加一些非标准关键字。

3. 运算符(Operator)

C 语言包含丰富的运算符，其由一个或多个字符组成。运算符与变量函数一起组成表达式，表示各种运算功能。按照不同的用途，运算符大致可分为 13 类，如表 3-2 所示。

表 3-1 C 语言的关键字

类 型	关 键 字					作 用
数据类型 (9 个)	int	float	double	char	void	数据类型的声明
	short		long			声明短整型和长整型
	signed		unsigned			声明有符号和无符号类型变量
自定义数据类型 (3 个)	struct					声明结构体数据类型
	union					声明联合数据类型
	enum					声明枚举数据类型
控制类型 (12 个)	if	else	switch	case	default	用于分支结构
	for	while	do...while			用于循环结构
	break					用于跳出当前循环或分支结构
	continue					结束本次循环，进入下一轮循环
	goto					无条件跳转语句
	return					返回到函数调用处
变量存储类型 (4 个)	auto					声明自动变量（一般情况下省略）
	extern					声明外部变量
	register					声明寄存器变量
	static					声明静态变量
其他类型 (4 个)	sizeof					计算数据类型长度
	const					声明只读变量
	typedef					用于给数据类型取别名等
	volatile					变量在程序执行中可被隐含改变

表 3-2 C 语言的运算符

序号	运算符类别	运算符
1	算术运算符	+ - * / %
2	关系运算符	> >= == < <= !=
3	逻辑运算符	! &&
4	赋值运算符	=
	复合赋值运算符	+= -= *= /= %= &= = ^= <<= >>=
5	自增和自减运算符	++ --
6	条件运算符	? :
7	强制类型转换运算符	()
8	指针和地址运算符	* &
9	计算字节数运算符	sizeof
10	下标运算符	[]
11	结构体成员运算符	->
12	位运算符	<< >> ^ & ~
13	逗号运算符	,

4. 分隔符(Separator)

分隔符用于分隔各个词法记号或程序正文的符号。在 C 语言程序中，空格、回车/换行、逗号等，在各自不同的应用场合起着分隔符的作用。

程序中相邻的关键字、标识符之间应由空格或回车/换行作为分隔符，逗号则用于相邻同类项之间的分隔；在定义相同类型的变量之间可用逗号分隔，在向屏幕输出的变量列表中，

各变量或表达式之间也用逗号分隔，如“`int a,b,c;`”。

关键字 `int` 和标识符 `a` 之间有空格，该空格为分隔符，如果没有空格，则为“`inta`”，系统会把它当成一个标识符，就无法实现定义变量的功能；标识符 `a` 和 `b`，`b` 和 `c` 之间都有一个逗号，它们同属于一种数据类型 `int`，因此可以共用关键字 `int` 进行定义，只需要用逗号分隔开即可。

5. 常量 (Constant)

在程序设计中会用到各种不同的数据，在这些数据中，有的在程序执行过程中不会发生变化，称为常量。

6. 注释符 (Comment)

C 语言的注释符是以“`/*`”开头，并以“`*/`”结尾的串。在“`/*`”和“`*/`”之间的内容为注释。以“`//`”开头的为单行注释。程序编译时，不对注释做任何处理。注释可出现在程序中的任何位置，用来向用户提示或解释程序的意义。在调试程序中对暂不使用的语句也可用注释符括起来，使编译程序跳过，不进行处理，待调试结束后再去掉注释符。在源程序中，使用注释是很有必要的，可有效增加程序的可读性。

3.2 常量和变量

大多数程序在输出之前都要执行一系列的计算，因此需要在程序执行过程中有一个临时存储数据的地方。C 语言中所说的变量，就是计算机内存中一个被命名的存储位置，它由一个或多个连续的字节组成。使用变量名来标识内存中的某个位置，再通过变量名就可读取该位置的数据，或在其中存储一个新数据。在程序中使用变量名，实际上引用的是内存中对应的某个存储位置。变量的值不是固定的，随时都可以改变，且次数不限。

与变量相同的是，常量也是程序使用的一个数据存储位置；不同的是，在程序运行期间，常量的值不能修改。

3.2.1 常量

C 语言有 5 种常量类型，包括：整型常量、实型常量、字符常量、字符串常量和符号常量。

- ① 整型常量有 3 种形式，包括：十进制整型常量、八进制整型常量、十六进制整型常量。
- ② 实型常量有 2 种形式，包括：十进制数形式和指数形式。
- ③ 字符常量代表 ASCII 码字符集里的一个字符，在程序中用单引号（`'`）括起来，如 `'A'`、`'a'`。
- ④ 字符串常量是用双引号（`"`）把一个或者多个字符括起来，这样构成的就是字符串常量。如 `"asdf"`、`"456"`、`"a"`，特别要注意的是，`"a"` 和 `'a'` 是 C 语言中两种完全不同类型的数据。

⑤ 符号常量是由 `#define` 命令定义的常量（宏常量）。`#define` 编译指令的准确含义是，命令编译器将源代码中所有标识符常量替换为替换文本。其效果与使用编辑器手工进行查找并

替换的相同。如`#define PI 3.1415926`, 编译预处理命令`#define`将PI定义成一个要被3.1415926取代的符号, 此时PI不是一个变量, 而是3.1415926的别名。在编译开始之前, 只要在程序的表达式中引用PI, 预处理器就会用`#define`指令中的值3.1415926来取代它。

通常, 符号常量名中的字母全部为大写, 这易于将其同变量名区分开来, 变量名中的字母一般为小写。程序员将所有的`#define`放在一起, 并将它们放在程序的开头。

宏常量也有其自身的缺点, 它被替换成文本后, 内存中会出现同一个替换文本的多个副本。为了弥补这个缺点, C语言推出`const`来定义常量。

`const`常量与变量定义形式类似, 加上`const`修饰, 可告诉编译器它的值是固定的, 不能被改变, 编译器会帮助检查、监督。

定义`const`常量, 会用到关键字`const`, 如“`const double PI = 3.1415926;`”。

`const`推出的初始目的是, 取代宏常量, 消除其缺点, 同时继承其优点。

在编译时, 由于`const`定义常量只是给出了对应的内存地址, 而不像`#define`给出的是替换文本。因此, `const`定义的常量在程序运行过程中只有一个副本, 而`#define`定义的常量在内存中有若干副本。

3.2.2 变量

变量是在程序执行过程中可以改变、可以赋值的量。在C语言中, 变量必须遵循“先定义, 后使用”的原则, 即每个变量在使用之前都要用变量定义语句将其定义为某种具体的数据类型。

1. 变量的四要素

变量类型、变量名、变量值和变量的指针(变量在内存中的地址)称为变量的四要素, 描述一个变量需通过四要素来完成。在C语言中, 变量命名必须遵循以下命名规则。

变量名是一种标识符, 只能由字母、数字和_(下画线)组合而成, 且必须以字母或_(下画线)开头。

C语言中的某些词(如`int`和`float`等)称为保留字, 具有特殊意义, 不能用作变量名。

C语言区分大小写, 因此变量`a`与变量`A`是两个不同的变量。

C99标准中规定变量名最多可以包含33个字符, 给变量取名最好能做到“见名知意”。对于由多个单词组合而成的变量名, 有多种风格, 其中采用较多的一种是, 使用下画线将单词分开, 如`teacher_name`, 如果不用下画线, 那么应该将第1个单词后面的每个单词都用大写字母表示, 如`teacherName`。程序员可以根据个人的喜好来选取命名风格。

变量值是指存储在变量中的数据, 准确地说就是存储在以变量名进行标识的存储单元中的值。

2. 变量的定义

在C语言程序中使用的每个变量都必须定义。要定义一个变量需要提供两方面信息: 变量名和类型。变量的类型将决定变量的存储结构, 以便编译程序为定义的变量分配存储空间。

变量的定义格式如下:

```
类型说明符 变量1, 变量2, ...;
```

其中，类型说明符是 C 语言中的一个有效的数据类型，如整型类型说明符 `int`、字符型类型说明符 `char` 等。

例如：

```
int  a,b,c;      /*定义 a,b,c 为整型变量*/
char ch;         /*定义 ch 为字符型变量*/
double x,y;      /*定义 x,y 为双精度实型变量*/
```

在 C 语言中，要求对所有用到的变量进行强制定义，也就是“先定义，后使用”，这样做的目的如下。

只有声明过的变量才可以在程序中使用，这使得变量名的拼写错误容易发现。例如，如果在定义部分写了“`int stu;`”，而在执行语句中错写成“`str=50;`”，则在编译时会检查出 `str` 未经定义，不作为变量名，输出“变量 `str` 未经声明”的信息，便于用户发现错误。

声明的变量属于确定的数据类型，编译系统可方便地检查变量进行运算的合法性。例如，整型变量 `a` 和 `b`，可以进行求余运算，得到 `a` 除以 `b` 的余数。如果将 `a` 和 `b` 指定为实型变量，则不允许进行求余运算，在编译时会给出有关的出错信息。

在编译时根据变量的数据类型可以为变量确定存储空间。例如，指定 `a` 和 `b` 为 `int` 型，那么使用 C 编译系统就会为变量确定 4 个字节的存储空间。

3.3 数据类型

在 C 语言中，数据是程序要处理的对象和结果，是程序设计中涉及和描述的主要内容，所以数据的分类非常重要。我们把程序能够处理的基本数据对象划分成一些集合，属于同一集合的各数据对象称为一种数据类型，每一种数据类型都具有同样的性质、表示形式、存储空间大小、构造特点等。

计算机执行程序时，组成程序的指令和程序操作的数据都必须存储于计算机的内存中。计算机硬件把被处理的数据分成一些类型，如定点数、浮点数等。CPU 对不同的数据类型提供不同的操作指令。在程序设计语言中，用数据类型来描述程序中的数据结构、数据表示范围、数据在内存中的存储分配等。C 语言的主要数据类型如图 3-1 所示。

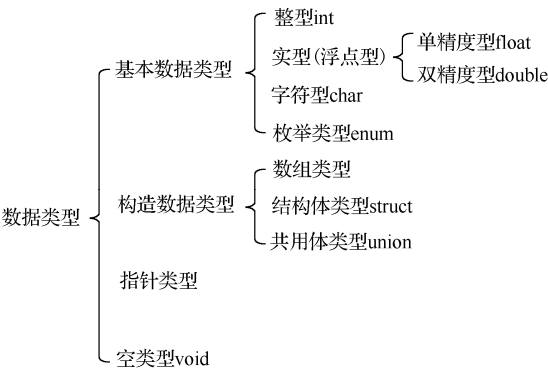


图 3-1 C 语言的主要数据类型

本章只介绍部分基本数据类型，其他数据类型将在后续章节继续介绍。

3.3.1 整型数据

在 C 语言中, 整型数据用于描述整数。整型数据包括整型常量、整型变量。整型常量就是整型常数, 整型数据在内存中以二进制补码形式存放。

1. 整型常量

C 语言的整型常量有三种表示形式。

(1) 十进制整型常量

十进制整型常量的表示与数学上的表示相同, 十进制整型常量前没有前缀, 由 0~9 的数字组成, 如 54、129、258。

(2) 八进制整型常量

八进制整型常量的表示形式是以数字 0 开头, 即以 0 作为八进制数的前缀, 由 0~7 的数字组成, 如 05、067、012。

(3) 十六进制整型常量

十六进制整型常量的表示形式以 0x 或 0X 作为前缀, 由数字 0~9、字母 A~F 或 a~f 组成, 如 0x58、0X2A。

整型常量的数据类型可以根据其值的范围来确定, 一个整数如果在 -2147483648~2147483647 范围内, 则认为是 int 型, 超过上述范围, 则认为是 long int 型, 可赋给 long int 型变量。一个 short int 型的常量也同时是一个 int 型常量, 可以赋给 int 型或 short int 型变量。常量无 unsigned 型, 但可将一个非负值且在取值范围内的整数赋给 unsigned 型变量。在一个整型常量后面加一个字母 l 或 L, 则认为是 long int 型常量。

2. 整型变量

(1) 整型变量的分类

整型变量以关键字 int 作为基本类型说明符, 另外配合 4 个类型修饰符, 用来改变和扩充基本类型的含义, 以便灵活应用。可用于基本型 int 的类型修饰符有 4 个, 分别为: long(长)、short(短)、signed(有符号)和 unsigned(无符号)。

这些修饰符与 int 可以组合成 6 种不同的整数类型, 也是 ANSI C 标准允许的整数类型。

- ① 有符号整型: [signed] int。
- ② 有符号短整型: [signed] short [int]。
- ③ 有符号长整型: [signed] long [int]。
- ④ 无符号整型: unsigned [int]。
- ⑤ 无符号短整型: unsigned short [int]。
- ⑥ 无符号长整型: unsigned long [int]。

注意: 在书写时, 如果既不指定为 signed, 也不指定为 unsigned, 则隐含意义为有符号 signed。由此可见, 有些修饰符是多余的, 如修饰符 signed。signed int、long int、signed long int 与 int 类型都是等价的。增加这些修饰符只是为了提高程序的可读性, signed 与 unsigned 对应, short 与 long 对应, 使用时可使程序看起来更加明了。

有符号整型数的存储单元的最高位是符号位(0 为正、1 为负), 其余为数值位。无符号整

型数的存储单元的全部二进制位用于存放数值本身，而不包含符号。表 3-3 为整型数据分类说明，列出了 32 位机上整型数据占用的内存空间及表示的数的精度，注意 int 和 long int 有着相同的取值范围。

表 3-3 整型数据分类说明

变量类型	类型说明符	别 称	取值范围	所需内存字节数
整型	int	无	$-2^{31} \sim 2^{31}-1$ ，-2147483648~+2147483647	4
短整型	short int	short	$-2^{15} \sim 2^{15}-1$ ，-32768~+32767	2
长整型	long int	long	$-2^{31} \sim 2^{31}-1$ ，-2147483648~+2147483647	4
无符号短整型	unsigned short int	unsigned short	$0 \sim 2^{16}-1$ ，0~65535	2
无符号整型	unsigned int	unsigned	$0 \sim 2^{32}-1$ ，0~4294967295	4
无符号长整型	unsigned long int	unsigned long	$0 \sim 2^{32}-1$ ，0~4294967295	4

(2) 整型变量的定义

整型变量的定义格式：

类型说明符 变量名；

32

【案例 3-1】 整型变量的定义。

```
#include<stdio.h>
int main()
{
    int num1,num2,sum;           /*定义整型变量 num1,num2,sum*/
    num1=3;                      /*为变量 num1 赋初值 3*/
    num2=5;
    sum=num1+num2;               /*计算 num1 与 num2 的和，并将结果赋给 sum*/
    printf("%d\n",sum);          /*输出变量 sum 的值*/

    return 0;
}
```

程序运行结果如图 3-2 所示。

说明：

定义变量时，类型说明符与变量名之间要有空格。说

明多个相同类型的变量时，各个变量之间用逗号分隔。

变量说明必须在变量使用之前，即“先定义，后使用”。

可以在定义变量的同时，对变量进行初始化。

(3) 整型数据的溢出

一个 int 型变量的最大允许值为 2147483647，如果再加 1，其结果不是 2147483648，而是 -2147483648，因为发生“溢出”。同样一个 int 型变量的最小允许值为-2147483648，如果再减 1，其结果不是-2147483649，而是 2147483647，也发生了“溢出”。

【案例 3-2】 整型数据的溢出。

```
#include<stdio.h>
int main()
```

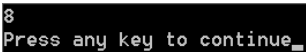


图 3-2 运行结果

```

{
    int a,b;

    a=2147483647;
    b=a+1;
    printf("\na=%d,a+1=%d",a,b);
    a=-2147483648;
    b=a-1;
    printf("\na=%d,a-1=%d\n",a,b);

    return 0;
}

```

程序运行结果如图 3-3 所示。

说明：

在 VC++ 6.0 环境中，一个整型变量只能容纳 -2147483648~2147483647 范围内的数，无法表示大于 2147483647 或小于 -2147483648 的数，否则就会发生“溢出”，但在运行时不会报错。它就像钟表一样，钟表的表示范围为 0~11，达到最大值后，又从最小数开始计数，因此，最大数 11 加 1 得不到 12，而得到 0。同样，最小数 0 减 1 也得不到 -1 而得到 11。在此例中我们可以看到最大的整型变量 2147483647 加 1 后得到的是最小值 -2147483648，而最小值 -2147483648 减 1 后得到最大值 2147483647。

C 语言的用法比较灵活，很多情况下会出现此类“副作用”，但系统却不给出“出错信息”，要靠程序员的细心检查和经验来保证结果的正确性。

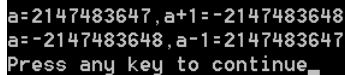


图 3-3 运行结果

3.3.2 实型数据

C 语言中实型数据又称为浮点型，根据其表示形式可分为实型常量和实型变量，主要用于表达和处理实数，实型变量只能存放实型常量。

(1) 实型常量

实型常量(浮点数)有以下两种表示形式。

① 十进制小数形式。由数字 0~9 和小数点组成(必须有小数点)。

例如，2.5、.36、-267.4280、0.0、123.都是合法的实型常量。

② 指数形式。由二进制数、阶码标志“e”或“E”及阶码(只能为整数，可以带符号)组成。其一般形式为：aEn(a 为十进制数，n 为二进制整数)，字母 e 或 E 之前必须有数字，之后的指数必须为整数。

例如，12.3e3、1.23E4 都是合法表示，都表示 12300。

一个实型常量可以有多种指数表示形式，但最好采用规范化的指数形式。所谓规范化的指数形式是指，在字母 e 或 E 之前的小数部分中，小数点左边应当有且只有一位非 0 数字。例如，123.456 可以表示为 123.456e0、12.3456e1、1.23456e2、0.123456e3 等，但只有 1.23456e2 可称为规范化的指数形式。用指数形式输出时，是按规范化的指数形式输出的。

C 编译系统将实型常量作为双精度型实数来处理。这样可以保证较高的精度，缺点是运算速度会降低。可以在实数的后面加字符 f 或 F，如 1.65f、654.87F，使编译系统按单精度型处理。

实型常量可以赋值给 float、double、long double 型变量，根据变量的类型截取实型常量中相应的有效数字。

(2) 实型变量

与整数存储方式不同，实型数据是按照指数形式存储的。系统将实型数据分为小数部分和指数部分，分别存放。

实型变量分为单精度型(float)、双精度型(double)。单精度型变量在内存中占 4 个字节(32 位)，双精度型变量在内存中占 8 个字节(64 位)。

实型变量说明的格式和书写规则与整型相同。

例如：

```
float x,y;      /*x,y 为单精度实型变量*/
double a,b,c;   /*a,b,c 为双精度实型变量*/
```

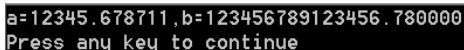
34

【案例 3-3】 实型数据的输出。

```
#include<stdio.h>
int main()
{
    float a;                /*说明变量 a 为单精度型*/
    double b;               /*说明变量 b 为双精度型*/

    a=12345.6789;           /*为 a 赋值*/
    b=0.123456789123456789e15; /*为 b 赋值*/
    printf("a=%f,b=%f\n",a,b); /*输出变量 a 和 b 的值*/
    return 0;
}
```

程序运行结果如图 3-4 所示。



```
a=12345.678711,b=123456789123456.780000
Press any key to continue
```

图 3-4 运行结果

说明：

程序为单精度型变量 a 和双精度型变量 b 分别赋值，并不经过任何运算就直接输出变量 a、b 的值。理想结果应该是按照原样输出，即 a=12345.6789，b=123456789123456.789。

但运行该程序后，实际输出结果是 a=12345.678711，b=123456789123456.780000。

由于实型数据的有效位是有限的，程序中变量 a 为单精度型，只有 7 位有效数字，所以输出的前 7 位是准确的，第 8 位及以后的数字 711 是无意义的。变量 b 为双精度型，可以有 15~16 位的有效位，所以输出的前 16 位是准确的，第 17 位及以后的数字 80000 是无意义的。由此可见，由于机器存储的限制，使用实型数据在有效位以外的数字将被舍去，因此可能会产生一些误差，需要我们在编程中注意。

为了进一步说明，可参考案例 3-4。

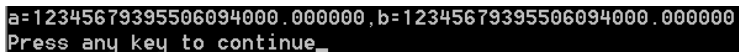
【案例 3-4】 实型数据的舍入误差。

```
#include<stdio.h>
int main()
{
    float a,b;

    a=1.23456789e19;           /*给实型变量 a 赋值*/
    b=a+2;                     /*将实型变量 a 的值加上 2 后赋给实型变量 b*/
    printf("a=%f,b=%f\n",a,b); /*以十进制小数形式输出实型变量 a 和 b 的值*/

    return 0;
}
```

程序运行结果如图 3-5 所示。



```
a=12345679395506094000.000000,b=12345679395506094000.000000
Press any key to continue_
```

图 3-5 运行结果

说明：

从运行结果看，程序运行时输出 b 的值与 a 相等。原因是 a 的值比 2 大很多，而一个单精度实型变量只能保证 7 位有效数字。运行程序得到 a 和 b 的值都是 12345679395506094000.000000。因此可以看到，前 7 位是准确的，后面都是不准确的，把数加在后几位上是无意义的。

由于实数存在舍入误差，使用时要注意以下几点：

- ① 不要试图用一个实数精确表示一个大整数，浮点数是不精确的；
- ② 实数一般不判断“相等”，而是判断接近或近似；
- ③ 避免直接将一个很大的实数与一个很小的实数相加、相减，否则会“丢失”小的数。

3.3.3 字符型数据

字符型数据包括字符常量和字符变量。

(1) 字符常量

字符常量是用单引号(')括起来的一个字符，如 'a'、'b'、'='、'+'、'?'都是合法的字符常量。

在 C 语言中，还规定了另外一类字符常量，它们以“\”开头，“\”称为转义字符。转义字符具有特定的含义，不同于字符原有的意义。例如，在前面各案例中 printf() 函数的格式串用到的“\n”就是一个转义字符，其意义是“换行”。

所有字符常量(包括可以显示的、不可显示的)均可以使用字符的转义表示法表示(ASCII 码表示)。转义字符主要用来表示那些用一般字符不便于表示的控制代码。

部分常用的转义字符及其含义如表 3-4 所示。

表 3-4 部分常用的转义字符及其含义

字 符	含 义	字 符	含 义
\n	换行(Newline)	\a	响铃报警(Alert or Bell)
\r	回车(但不换行, Carriage Return)	\"	一个双引号(Double Quotation Mark)
\0	空字符, 通常用作字符串结束标志(Null)	\'	一个单引号(Single Quotation Mark)
\t	水平制表(Horizontal Tabulation)	\\	一个反斜线(Backslash)
\v	垂直制表(Vertical Tabulation)	\?	一个问号(Question Mark)
\b	退格(Backspace)	\ddd	1~3 位八进制 ASCII 码值所代表的字符
\f	走纸换页(Form Feed)	\xhh	1~2 位十六进制 ASCII 码值所代表的字符

说明:

转义字符大致分为以下三类。

- ① 第一类是“\”后跟一个字母, 表示某些控制字符。例如, “\r”表示回车, “\b”表示退格等。
- ② 第二类是表示单引号、双引号和反斜线, 写为“\'”、“\”、“\\”。
- ③ 第三类是“\ddd”和“\xhh”这两种表示法, 可以表示 C 语言字符集中的任何一个字符。ddd 和 xhh 分别为八进制和十六进制的 ASCII 代码。例如, “\101”表示字符 A, “\x41”也表示字符 A, “\102”表示字符 B, “\134”表示“反斜线”等。

【案例 3-5】 转义字符的使用。

```
#include<stdio.h>
int main()
{
    int a,b,c;
    a=1234567;b=8;c=9;
    printf("%d\n\t%d %d\n %d %d\t\b%d",a,b,c,a,b,c);
    return 0;
}
```

程序运行结果如图 3-6 所示。

说明:

程序在第 1 列输出 a 的值 1234567 之后执行“\n”, 回车换行; 接着执行“\t”, 跳到下一制表位置(设制表位置间隔为 8), 再输出 b 的值 8; 空 1 格再输出 c 的值 9 后又是“\n”, 再回车换行; 再空 1 格后又输出 a 的值 1234567; 再空 1 格又输出 b 的值 8; 再次跳到下一制表位置, 但执行转义字符“\b”, 退格, 又退回上一制表位置, 在 8 的位置再输出 c 的值 9。

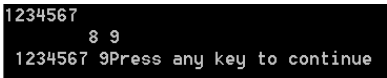


图 3-6 运行结果

(2) 字符变量

字符变量用于存放字符常量, 即 1 个字符变量可存放 1 个字符, 所以 1 个字符变量占用 1 个字节的内存容量。说明字符变量的关键字是 char, 使用时只需在说明语句中指明字符型数据类型和相应的变量名即可。

例如:

```
char c1,c2; /*说明 c1,c2 为字符变量*/
c1= 'A'; /*为 c1 赋值为字符常量 A*/
c2= 'a'; /*为 c2 赋值为字符常量 a*/
```

(3) 字符型数据在内存中的存储及使用

字符型数据在内存中是以字符的 ASCII 码的二进制形式存放的, 占用 1 个字节, 如图 3-7 所示。

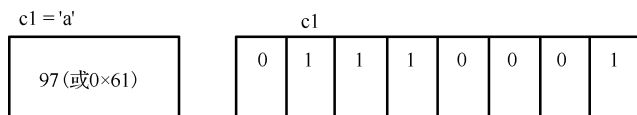


图 3-7 数据在内存中的存储形式

从图 3-7 可以看出字符数据以 ASCII 码存储的形式与整数的存储形式类似, 这使得字符型数据和整型数据可以通用 (0~255 范围内的无符号数或 -128~127 范围内的有符号数), 具体表现为如下几点。

- ① 可以将整型常量赋值给字符型变量, 也可以将字符型常量赋值给整型变量。
- ② 可以对字符型数据进行算术运算, 相当于对它们的 ASCII 码进行算术运算。
- ③ 一个字符型数据既可以字符形式输出 (ASCII 码对应的字符), 也可以整数形式输出 (直接输出 ASCII 码)。
- ④ 字符型数据和整型数据之间可以通用, 但是字符型只占 1 个字符, 即如果作为整数使用, 只能存放 0~255 范围内的无符号数或有符号数。

【案例 3-6】 字符变量的使用。

```
#include<stdio.h>
int main()          /*字符'a'的各种表达方法*/
{
    char c1='a';
    char c2='\x61';    /*\x61 为转义字符*/
    char c3='\141';    /*\141 为转义字符*/
    char c4=97;
    char c5=0x61;      /*0x61 为十六进制数, 相当于十进制数 97*/
    char c6=0141;      /*0141 为八进制数, 相当于十进制数 97*/
    printf("\nc1=%c,c2=%c,c3=%c,c4=%c,c5=%c,c6=%c\n",c1,c2,c3,c4,c5,c6);
    /*以字符形式输出*/
    printf("\nc1=%c,c2=%d,c3=%d,c4=%d,c5=%d,c6=%d\n",c1,c2,c3,c4,c5,c6);
    /*以十进制整数形式输出*/
    return 0;
}
```

程序运行结果如图 3-8 所示。

```
c1=a,c2=a,c3=a,c4=a,c5=a,c6=a
c1=a,c2=97,c3=97,c4=97,c5=97,c6=97
Press any key to continue
```

图 3-8 运行结果

【案例 3-7】 大、小写字母的转换。

```
#include<stdio.h>
int main()
```

```

{
    char  c1,c2,c3,c4;

    c1='a';
    c2=c1+1;    /*c1 的 ASCII 码值 97 加 1 后赋给 c2*/
    c3=c1-32;    /*c1 的 ASCII 码值 97 减 32 后赋给 c3*/
    c4=c2-32;    /*c2 的 ASCII 码值 98 减 32 后赋给 c4*/
    printf("\n %c,%c,%c,%c\n",c1,c2,c3,c4);    /*按字符形式输出各变量的值*/
    printf("%d,%d,%d,%d\n",c1,c2,c3,c4);    /*按 ASCII 码形式输出各变量的值*/

    return 0;
}

```

运行结果如图 3-9 所示。

说明：

本程序的作用是将两个小写字母 a 和 b 转换成大写字母 A 和 B。从 ASCII 码表中可以看到每一个小写字母比对应的大写字母的 ASCII 码大 32。本例还反映出允许字符型数据与整型数据直接进行算术运算，运算时字符型数据用 ASCII 码值参与运算，字符型数据可以以整数形式输出，输出的是其 ASCII 码值。

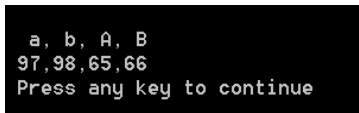


图 3-9 运行结果

38

(4) 字符串常量

字符串常量是用一对双引号(" ")括起来的字符序列。这里的双引号仅起到字符串常量的边界符的作用，它并不是字符串常量的一部分，如"Hello world.\n"、"ABC"、" "、"a"。

C 语言规定在每个字符串的结尾加一个“字符串结束标志”，以便系统据此判断字符串是否结束。C 语言规定以'\0' (ASCII 码为 0 的字符) 作为字符串的结束标志。

例如，"CHINA"在内存中的存储形式如图 3-10 所示(存储长度=6)。

C	H	I	N	A	\0
---	---	---	---	---	----

图 3-10 "CHINA"在内存中的存储形式

可见，字符常量与字符串常量的区别有两个：从形式上看，字符常量是用单引号括起来的单个字符，而字符串常量是用双引号括起来的一串字符；从存储方式看，字符常量在内存中占 1 个字节，而字符串常量除每个字符各占 1 个字节外，其字符串结束标志'\0'也要占 1 个字节。例如，字符常量'a'占 1 个字节，而字符串常量"a"占 2 个字节。值得注意的是，尽管'\0'对于字符串是必不可少的，但是在书写时并不需要将它写在字符串的后面，'\0'由 C 语言编译器自动处理。

如果字符串常量中出现双引号，则要用反斜线“\”将其转义，取消原有边界符的功能，使之仅作为双引号字符使用。例如，要输出字符串 He says: "How do you do."，应写成：

```
printf("He says:\"How do you do.\\");
```

C 语言没有专门的字符串变量，如果想将 1 个字符串存放在变量中，可以使用字符数组(即用 1 个字符数组来存放 1 个字符串，数组中每 1 个元素存放 1 个字符)。

3.4 运算符与表达式

C 语言拥有丰富的运算符，基本运算符在大多数编程语言中都有，但是 C 语言不止包括这些运算符，本节将介绍最为常用的运算符。C 语言的运算符具有不同的优先级 (Precedence) 和结合性 (Associativity)。在表达式中，各运算量参与运算的先后顺序不仅要遵守运算符优先级的规定，还要受运算符结合性的制约，以便确定是自左向右进行运算，还是自右向左进行运算。这种结合性其他高级语言的运算符所没有的，因此也增加了 C 语言的复杂性。

(1) 运算符的分类

① 按在表达式中与运算对象的关系(连接运算对象的个数)可以分为以下 3 类。

- 单目运算符：一个运算符连接一个运算对象。
- 双目运算符：一个运算符连接两个运算对象。
- 三目运算符：一个运算符连接三个运算对象。

② 按它们在表达式中起到的作用又可以分为以下几种。

- 算术运算符：包括+、-、*、/ 和%。
- 自增、自减运算符：包括++和--。
- 赋值运算符与复合赋值运算符：包括=、+=、-=、*=、/=、%=、<<=、>>=、^=、&=和|=。
- 关系运算符：包括<、<=、>、>=、==和!=。
- 逻辑运算符：包括&&、||和!。
- 位运算符：包括~、|、&、<<、>>和^。
- 条件运算符：包括?:。
- 逗号运算符：包括,。
- 其他：包括*、&、()、[]、->和 sizeof。

(2) 运算符的优先级和结合性

优先级：指同一个表达式中不同运算符进行计算时的先后次序。

结合性：结合性是针对同一优先级的多个运算符而言的，是指同一个表达式中相同优先级的多个运算应遵循的运算顺序。

左结合性(自左向右结合方向)：运算对象先与左边的运算符结合。

右结合性(自右向左结合方向)：运算对象先与右边的运算符结合。

数学中的四则运算，乘、除的优先级高于加、减，而乘、除之间和加、减之间是同级运算，其结合性均为左结合性。

例如， $a-b+c$ ，到底是 $(a-b)+c$ 还是 $a-(b+c)$ ？由于+、-运算优先级别相同，结合性为“自左向右”，也就是说 b 先与左边的 a 结合，所以 $a-b+c$ 等价于 $(a-b)+c$ 。

C 语言的运算符也同样具有运算的优先级和结合性。

以下将介绍其中重要的几种运算符与表达式。

3.4.1 算术运算符与算术表达式

C 语言提供的基本算术运算符 (Arithmetic Operators) 及相关说明如表 3-5 所示。

表 3-5 算术运算符及相关说明

运 算 符	含 义	类 型	表达式示例
+	加法或取正值运算	双目、单目运算符	x+y、+6
-	减法或取负值运算	双目、单目运算符	x-y、-6
*	乘法运算	双目运算符	x*y
/	除法运算	双目运算符	x/y
%	求余运算(取模运算)	双目运算符	4%3

虽然算术运算符与数学中的数值运算很相似，但仍存在差别，使用时需注意以下几点。

① 两个整数相除的结果为整数，如 5/3 的结果为 1，舍去小数部分。例如，6/4 与 6.0/4 运算结果的值是不同的，6/4 的值为整数 1，而 6.0/4 的值为实数 1.5。这是因为，当其中一个操作数为实数时，整数与实数运算的结果为 double 型。

② 如果参加+、-、*、/运算的两个数有一个为实数，则结果为 double 型，因为所有实数都按 double 型进行计算。

③ 求余运算符%，要求两个操作数必须都为整型，结果为两数相除所得的余数。一般情况下，余数的符号与被除数符号相同。例如，-8%5=-3、8%-5=3。

④ +、-运算符既具有单目运算符的取正值和取负值功能，又具有双目运算符的功能。作为单目运算符使用时，其优先级高于双目运算符。

算术表达式是指用算术运算符和括号将运算对象(操作数)连接起来的，符合 C 语言语法规则的式子。运算对象包括常量、变量、函数等。下面是一个合法的算术表达式：

```
6+a/b*5+8%3+'a'
```

3.4.2 自增、自减运算符与表达式

++是自增运算符，它的作用是使变量的值增加 1。--是自减运算符，它的作用与自增运算符相反，即让变量的值减 1。它们均为单目运算符。

例如：

```
k++;          /*相当于 k=k+1*/
k--;          /*相当于 k=k-1*/
```

自增、自减运算符既可作前缀运算符，也可作后缀运算符。无论是前缀运算符还是后缀运算符，对于变量本身来说，都是自增 1 或自减 1，具有相同的效果；但对表达式来说，对应值却不同。采用前缀形式，在计算表达式的值时，取变量增、减变化后的值，即新值；采用后缀形式，则在计算表达式值时，取变量增、减变化前的值，即旧值。两种形式中，表达式的值相差 1。

说明：

后缀(如 k++)是先使用变量的值，再使变量自增 1；前缀(如++k)是先使变量增 1，再使用该变量的值。--运算类似。

例如：

```
int k,m,i=4,j=4;
k=i++;
m=++j;
```

该程序段执行完后，i 和 j 的值均为 5，而 k 的值为 4(后缀形式，取 i 的旧值)，m 的值为 5(前缀形式，取 j 的新值)。

自增、自减运算符的运算对象只能为变量，不能为常量或表达式，因为常量的值是不允许改变的，表达式的值实际上也相当于一个常量，它们都不能放在赋值号的左边。

【案例 3-8】 自增 1 和自减 1 运算符的使用。

```
#include<stdio.h>
int main()
{
    int i=5;
    printf("%d\n",++i);          /*i 加 1 后输出 6,i=6*/
    printf("%d\n",--i);          /*i 减 1 后输出 5,i=5*/
    printf("%d\n",i++);          /*输出 i 为 5 后再加 1(i 为 6)*/
    printf("%d\n",i--);          /*输出 i 为 6 后再减 1(i 为 5)*/
    printf("%d\n",-i++);         /*输出-5 后再加 1(i 为 6)*/
    printf("%d\n",-i--);         /*输出-6 后再减 1(i 为 5)*/
    printf("%d\n",i);            /*输出 i 的值 5*/
    return 0;
}
```

程序运行结果如图 3-11 所示。

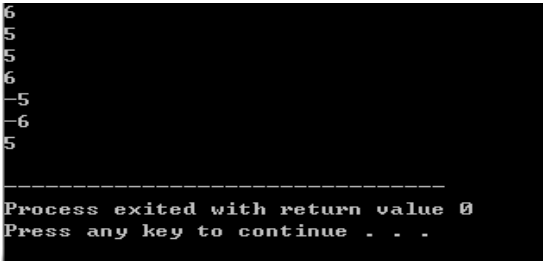


图 3-11 运行结果

3.4.3 关系运算符与关系表达式

关系运算符(Relational Operators)用来比较两个数据的大小关系，运算的结果为“真”，用 1 表示；结果为“假”，用 0 表示。C 语言提供了 6 种关系运算符，如表 3-6 所示。

表 3-6 关系运算符及相关说明

运算符	功 能	关系表达式示例	运算结果
<	小于	6<5	0
>	大于	6>5	1
<=	小于等于	6<=5	0

续表

运算符	功 能	关系表达式示例	运算结果
>=	大于等于	6>=5	1
==	等于	6==5	0
!=	不等于	6!=5	1

关系运算实质上就是比较运算。用关系运算符将两个操作数连接起来组成的表达式称为关系表达式。关系表达式通常用于表达一个判断条件的真与假。一个判断条件的真与假分别代表判断条件成立与不成立。C 语言并未提供逻辑型数据类型，那么，在 C 语言中如何表示判断条件的真与假呢？

在 C 语言中，用非 0 值表示“真”，用 0 值表示“假”。不管表达式为何种类型，只要表达式的值为 0，就表示其值为假，即条件不成立；只要表达式的值为非 0 值(也包括负数)，就表示其值为真，即条件成立。

C 语言对真、假值的判断策略使程序在判断条件方面更加灵活，只要是合法的，任何类型的 C 表达式都可作为判断条件，从而可编写出效率更高的子程序。

【案例 3-9】 求两个整数的最大值，并输出。

42

```
#include<stdio.h>
int main()
{
    int a,b,max;

    scanf("%d,%d",&a,&b);          /*输入 a、b 的值*/
    if(a>b)max=a;                  /*如果 a 大于 b，把 a 赋给 max*/
    else max=b;                    /*否则把 b 赋给 max*/
    printf("max=%d\n",max);        /*输出 max 的值*/

    return 0;
}
```

程序运行结果如图 3-12 所示。

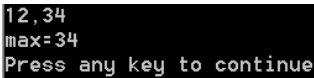


图 3-12 运行结果

3.4.4 逻辑运算符与逻辑表达式

为了表示数学中的不等式，如“ $a>b>c$ ”，C 语言引入了逻辑运算符(Logic Operators)。C 语言提供 3 种逻辑运算符，即：&&(逻辑与)、||(逻辑或)和!(逻辑非)，如表 3-7 所示。

表 3-7 逻辑运算符与逻辑表达式及相关说明

运算符	类 型	功 能	逻辑表达式示例	运算结果
!	单目	逻辑非	!6、!0	0、1
&&	双目	逻辑与	2&&3、0&&1	1、0
	双目	逻辑或	2 3、0 1	1、1

“逻辑与”运算的特点是：只有两个操作数都为真，结果才为真；只要有一个为假，结果

就为假。因此，当要表示两个条件必须同时成立时，可用“逻辑与”运算符连接这两个条件。

“逻辑或”运算的特点是：只要有一个操作数为真，结果就为真；只有两个操作数都为假，结果才为假。因此，当需要表示“或者……或者……”这样的条件时，可用“逻辑或”运算符连接这两个条件。

用逻辑运算符连接操作数组成的表达式称为逻辑表达式。逻辑表达式的值(或称逻辑运算的结果)只有真和假两个值。C标准规定，用1表示真，用0表示假。但是在判断一个数值表达式(不一定是逻辑表达式)的真、假时，如if和while语句中用于表示控制条件的表达式，由于一个任意数值表达式的值不只局限于0、1两种情况，因此以表达式的值是非0还是0来判断真、假。

逻辑表达式求解时，只有必须执行下一个逻辑运算符才能求出表达式的值时，才执行该运算符。

对于“逻辑与”，只有当运算符左边的值为真时，才计算运算符右边的值。例如：

```
int a=1,b=1,c=1,d=1,m=1,n=1;    n=(a=2)&&(b=2);    /*执行后 n=1,a=2,b=2*/
m=(a=2)&&(b=0)&&(c=0);            /*执行后 m=0,a=2,b=0,c=1*/
```

对于“逻辑或”，只有当运算符左边的值为假时，才计算运算符右边的值。例如：

```
int a=1,b=1,c=1,d=1,m=1,n=1;
n=(a=2)|| (b=2);                /*n=1,a=2,b=1*/
m=(a=0)|| (b=2)|| (c=2);        /*m=1,a=0,b=2,c=1*/
```

3.4.5 赋值运算符与赋值表达式

赋值运算符(Assignment Operators)包括一般赋值运算符和复合赋值运算符。

1. 一般赋值运算符与表达式

一般赋值运算符的含义是将一个数据赋给一个变量，虽然书写形式与数学中的等号相同，但两者的含义是截然不同的。一般形式如下：

变量名 = 表达式

一般赋值运算符：“=”为双目运算符，遵循右结合性。

一般赋值表达式：由赋值运算符组成的表达式称为赋值表达式。例如：

```
a=6;
```

赋值表达式的求解过程如下：

- ① 先计算赋值运算符右侧的“表达式”的值；
- ② 将赋值运算符右侧“表达式”的值赋值给左侧的变量；
- ③ 整个赋值表达式的值就是被赋值变量的值。

赋值的含义：将赋值运算符右边的表达式的值存放到左边变量名标识的存储单元中。

2. 复合赋值运算符与表达式

为了简化程序的书写并提高编译效率，在赋值运算符“=”之前加上其他运算符，可构成复合赋值运算符，如+=、-=、*=、/=、%=、<<=、>>=、&=、|=和^=(共10种)。

复合赋值运算符均为双目运算符，遵循右结合性。

复合赋值运算符构成赋值表达式的一般格式如下：

变量名 复合赋值运算符 表达式

功能：对“变量名”和“表达式”进行复合赋值运算符规定的运算，并将运算结果赋值给复合赋值运算符左边的“变量名”。

复合赋值运算的作用等价于：

变量名=变量名 运算符 表达式

即先将变量和表达式进行指定的复合运算，然后将运算的结果赋给变量。

例如， $a*=3$ 等价于 $a=a*3$ ， $a/=b+5$ 等价于 $a=a/(b+5)$ 。

注意：一般赋值运算符、复合赋值运算符的优先级比算术运算符低，如“ $a*=b+5$ ”与“ $a=a*b+5$ ”是不等价的，它实际上等价于“ $a*(b+5)$ ”，这里的括号是必需的。

一般赋值表达式也可以包含复合赋值运算符，如“ $a+=a-a*a$ ”。

如果 a 的初值为 12，此表达式的求解步骤如下：

- ① 先进行“ $a-=a*a$ ”的运算，它相当于 $a=a-a*a$ ， $a=12-144=-132$ ；
- ② 再进行“ $a+=-132$ ”的运算，相当于 $a=a+(-132)=-132-132=-264$ 。

【案例 3-10】 复合赋值运算符的使用。

```
#include<stdio.h>
int main()
{
    int  a=5,b=6,c=2,d=7,x;

    a+=b*c;
    b-=c/b;
    printf("%d,%d,%d\n",a,b,c*=2*(a-b));
    d%=a;
    printf("x=%d\n",x=a+b+c+d);

    return 0;
}
```

程序运行结果如图 3-13 所示。

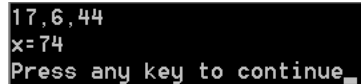


图 3-13 运行结果

3.4.6 逗号运算符与逗号表达式

C 语言提供了一种特殊的运算符，即逗号运算符(又称顺序求值运算符)。用它将两个或多个表达式连接起来，表示顺序求值(顺序处理)。

用逗号连接起来的表达式，称为逗号表达式。逗号表达式的一般形式如下：

表达式 1，表达式 2，…，表达式 n

逗号表达式的求解过程自左向右，即求解表达式 1，求解表达式 2，…，求解表达式 n 。整个逗号表达式的值是表达式 n 的值。例如，逗号表达式“ $3+5,6+8$ ”的值为 14。

查运算符优先级表可知,“=”运算符的优先级高于“,”运算符(事实上逗号运算符级别最低),结合性为自左向右。

【案例 3-11】 求逗号表达式的值。

```
#include<stdio.h>
int main()
{
    int x,a;

    x=(a=3,6*3);          /*把逗号表达式的值赋给变量 x, a=3, x=18*/
    printf("%d,%d\n",a,x);
    x=a=3,6*a;            /*a=3, x=3, 整个逗号表达式的值为 18*/
    printf("%d,%d\n",a,x);

    return 0;
}
```

程序运行结果如图 3-14 所示。

逗号表达式主要用于将若干表达式“串联”起来,表示一个顺序的操作(计算)。在许多情况下,使用逗号表达式的目的只是想分别得到各个表达式的值,而并非一定需要得到和使用整个逗号表达式的值。

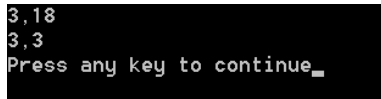


图 3-14 运行结果

并不是任何地方出现的逗号都是逗号运算符。如在变量说明中,函数参数表中逗号只用作各变量之间的分隔符,例如:

```
printf("%d,%d,%d",a,b,c);
```

其中“a,b,c”并不是一个逗号表达式,它是 printf 函数的 3 个参数,参数间用逗号分隔。如果改写为:

```
printf("%d,%d,%d", (a,b,c),b,c);
```

则“(a,b,c)”是一个逗号表达式,它的值等于 c 的值。括号内的逗号不是参数间的分隔符,而是逗号运算符。括号中的内容是一个整体,作为 printf() 函数的一个参数。C 语言的表达式类型丰富,运算符功能强大,因此 C 语言使用灵活,适应性强。

3.4.7 条件运算符与条件表达式

条件运算符“?:”是 C 语言中唯一的三目运算符,即需要三个数据或表达式构成条件表达式,其一般形式如下:

```
表达式 1?表达式 2:表达式 3
```

条件表达式的操作过程为:如果表达式 1 成立,则表达式 2 的值就是此条件表达式的值;否则,表达式 3 的值就是此条件表达式的值。例如:

```
max=a>b?a:b;
```

说明:

① “?:”运算符的优先级高于逗号运算符和赋值运算符。

对于“ $x=a>?a:b;$ ”，赋值号右边不必加小括号，因为“ $?:$ ”的优先级高于“ $=$ ”。

② “ $?:$ ”运算符的优先级低于算术运算符和比较运算符。

对于“ $x=a-3>b+2?a+1:b-3;$ ”也不必加小括号，运算符的运算顺序为“ $a-3$ ”、“ $b+2$ ”、“ $a+1$ ”、“ $b-3$ ”、“ $>$ ”、“ $?:$ ”、“ $=$ ”。

③ 条件运算符的结合方向为“自右向左”。

④ 条件表达式中的表达式类型可以不一样。

例如，以下语句是正确的：

```
x>y?3:1.24; ch>=a?printf("ch>a"):printf("ch<=a");
```

【案例 3-12】 条件表达式的使用，利用条件表达式求 a、b、c 中的最大者。

```
#include<stdio.h>
int main()
{
    float a,b,c,max;
    scanf("%f,%f,%f",&a,&b,&c);
    max=a>b?a>c?a:c:b>c?b:c;
    printf("max=%f\n",max);

    return 0;
}
```

程序运行结果如图 3-15 所示。

读者一定会认为程序中的语句“ $\text{max}=a>b?a>c?a:c:b>c?b:c;$ ”太难理解。事实上，这句等价于“ $\text{max}=a>b?(a>c?a:c):(b>c?b:c);$ ”，即若 a 大于 b，则取 a、c 中的较大者，否则取 b、c 中的较大者。去掉括号不影响结果，这正体现了条件表达式“自右向左”的结合性，当然，加上括号更好理解。



```
12.8,33
max=33.000000
Press any key to continue_
```

图 3-15 运行结果

3.4.8 sizeof 运算符与 sizeof 表达式

sizeof 运算符是 C 语言中一种比较特殊的单目运算符，也是一个非常有用的运算符。它以字节形式给出了其操作数的存储大小。操作数可以是一个表达式或括在括号内的类型名。操作数的存储大小由操作数的类型决定。其一般形式为：

```
sizeof(变量或数据类型)
```

【案例 3-13】 理解运算符 sizeof 的用法。

```
#include<stdio.h>
int main()
{
    int a=5;
    printf("%d,%d,%d,%d\n",sizeof(a),sizeof(int),sizeof(float),sizeof(char));
    return 0;
}
```

程序运行结果如图 3-16 所示。


```
4.4.4.1
-----
Process exited with return value 0
Press any key to continue . . .
```

图 3-16 运行结果

3.5 数据类型的转换

C 语言中不同数据类型的取值范围不同，进行混合运算时搞清楚运算结果的类型是非常重要的。

1. 自动类型转换

自动类型转换也叫隐式转换，是计算机按照默认规则自动进行的。C 语言规定，不同类型的数据在进行混合运算之前先转换成相同的类型，然后再进行运算，转换的规则如下。

① 所有的 `char` 型和 `short` 型一律先转换为 `int` 型，所有的 `float` 型先转换为 `double` 型再参加运算。

② 当算术运算符“+”、“-”、“*”、“/”、“%”两边的数据类型不一致时，就高不就低。这里的“高”和“低”是指数据所占存储空间的大小。例如：

```
int x=1; float y=x+1.5;
```

因为 `y` 要先转换为占 8B 的 `double` 型，而 `int` 型的 `x` 只占 4B，故 `x` 也要转换为 `double` 型之后才能进行加 1.5 的运算。

③ 当赋值号两边的类型不一致时，右向左看齐。例如：

```
int i; float x=26.789; i=x;
```

`i` 的结果为 26。

④ 当函数定义时的形式参数和调用时的实际参数类型不一致时，实际参数自动转换为形式参数的类型。

2. 强制类型转换

强制类型转换也称为显式转换。C 语言提供一种“强制类型转换”运算符，用它可以强迫表达式的值转换为某一特定类型。一般形式如下：

```
(类型说明符)表达式
```

强制类型转换最主要的用途有以下几方面。

① 满足一些运算符对类型的特殊要求。

例如，取余运算要求“%”两侧的数据类型必须为整型。“17.5%9”的表示方法是错误的，但“(int)17.5%9”就是正确的。

另外，C 语言的一些库函数(如 `malloc`)的调用结果是空类型(`void`)，则必须根据需要进行类型的强制转换，否则调用结果就无法利用。

② 防止整数在进行乘、除运算时小数部分丢失。

【案例 3-14】 强制类型转换的应用。

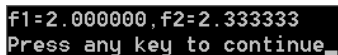
```
#include<stdio.h>
int main()
{
    int x=7,y=3;
    float f1,f2;

    f1=x/y;
    f2=(float)x/y;
    printf("f1=%f,f2=%f\n",f1,f2);

    return 0;
}
```

程序运行结果如图 3-17 所示。

在 C 语言中,用得最频繁的库函数就是 printf(),但无论是把一个整型数按照“%f”输出,还是把一个实型数按照“%d”输出,结果都是错误的。



```
f1=2.000000,f2=2.333333
Press any key to continue_
```

图 3-17 运行结果

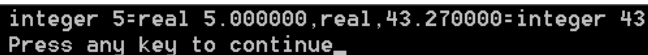
【案例 3-15】 在 printf() 函数中使用强制类型转换。

```
#include<stdio.h>
int main()
{
    int x=5;
    float y=43.27;

    printf("integer %d=real %f,real,%f=integer %d\n",x,(float)x,y,(int)y);

    return 0;
}
```

程序运行结果如图 3-18 所示。



```
integer 5=real 5.000000,real,43.270000=integer 43
Press any key to continue_
```

图 3-18 运行结果

可见采用强制转换,才能输出正确结果。

3.6 本章小结

本章主要介绍 C 语言中的一些基本概念,如常量、变量、标识符等,同时详细介绍了 C 语言中的几种基本数据类型、常用的运算符及利用这些运算符来构成相应表达式的一些规则。

对于变量必须“先定义,后使用”。变量的属性包括:类型、名字、值和地址。不同类型的变量占用内存空间的大小、数据的存储形式、合法的取值范围及可参与的运算类型都是不

同的。即便是同类型变量在不同的系统或平台中,其所占内存的字节数也不尽相同,因此不要对变量所占的内存空间的字节数想当然,要使用 `sizeof` 获得变量或者数据类型占用内存空间的字节数,以保证程序良好的可移植性。

一些常用运算符的优先级与结合性归纳如表 3-8 所示。

表 3-8 常用运算符的优先级与结合性

优先级顺序	运算符种类	附 加 说 明	结合方向
1	单目运算符	逻辑非! 按位取反~ 求负- ++ -- 类型强制转换等	右→左
2	算术运算符	* / % 高于 + -	左→右
3	关系运算符	< <= > >= 高于 == !=	左→右
4	逻辑运算符	除逻辑非之外, && 高于	左→右
5	赋值运算符	= += -= *= /= %= &= ^= = <<= >>=	右→左
6	逗号运算符	,	左→右

千万不要死记硬背这些内容,所谓熟能生巧,用则自然能熟,熟则自然能生巧。即使不能熟记 C 语言运算符的优先级与结合性,只要将需要先计算的表达式用括号括起来即可保证运算的正确性。

由于本书的篇幅有限,部分运算符及表达式未能详细介绍(如位运算符与位运算表达式等),请读者自行学习。



3.7 习题

一、单选题

- 下列字符序列中,不可用作 C 语言标识符的是()。
 - abc123
 - no.1
 - _123_
 - _ok
- 正确的 C 语言标识符是()。
 - _buy_2
 - 2_buy
 - ?_buy
 - buy?
- 请选出可用作 C 语言用户标识符的一组标识符()。

A. void	B. a3_b3	C. For	D. 2a
define	_123	-abc	DO
WORD	IF	Case	sizeof
- 下列符号中,不属于转义字符的是()。
 - \\
 - \0xAA
 - \t
 - \0
- 不属于 C 语言关键字的是()。
 - int
 - break
 - while
 - character
- 属于 C 语言提供的合法关键字的是()。
 - Float
 - signed
 - integer
 - Char
- 以下不能定义为用户标识符的是()。
 - scanf
 - Void
 - _3com_
 - int

8. 以下选项中, 合法的用户标识符是()。

- A. long B. _2abc C. 3dmax D. A.dat

9. 以下选项中, 合法的实型常数是()。

- A. 5E2.0 B. E-3 C. 2E0 D. 1.3E

10. 已知大写字母 A 的 ASCII 码值是 65, 小写字母 a 的 ASCII 码是 97, 则用八进制数表示的字符常量 '\01' 是()。

- A. 字符 A B. 字符 a C. 字符 c D. 非法的常量

11. 以下选项中, 合法转义字符的选项是()。

- A. '\ ' B. '\018' C. 'xab' D. '\abc'

12. 以下选项中, 正确的字符常量是()。

- A. "F" B. "\ " C. 'W' D. "

13. 在有字符型、整型、实型常数的表达式运算中, 其最后结果的类型是()。

- A. char 型 B. long 型 C. float 型 D. double 型

14. 字符串 "I am student" 在内存中占用的字节数是()。

- A. 12 个 B. 13 个 C. 14 个 D. 15 个

15. Turbo C 2.0 中, 若定义 unsigned long b, 则变量 b 在内存中分配的字节数是()。

- A. 1 个 B. 2 个 C. 4 个 D. 8 个

16. 下面表达式结果为 3 的是()。

- A. (-7)%4 B. (-7.0)%4.0 C. 7%(-4) D. 7.0%4.0

17. 若有 "int a,b;" , 下面正确的表达式是()。

- A. 7.0%3.0 B. (a+b)++ C. 7++ D. a+'a'

18. 设有 "int a=4,b=-7;" , 表达式 (int) (a+b)/2.0+a/b 的值是()。

- A. 4 B. -7 C. -1 D. -1.5

19. 设有 int a=2, 表达式 (a<=2/a)>>1 的值是()。

- A. 1 B. 2 C. 4 D. 8

20. 设有 int a=3, a+=a-=a*a 的值是()。

- A. 18 B. 9 C. -12 D. 3

21. 设有 "int a=3,b=-4,c=5;" , 表达式 a++-c+(++b) 的值是()。

- A. -7 B. -5 C. -4 D. -3

22. 运算符 ++、*、>>、=、,、sizeof 的优先级由高到低排序是()。

- A. ++、*、>>、=、,、sizeof B. *、++、>>、=、,、sizeof
C. ++、*、sizeof、=、,、>> D. sizeof、++、*、>>、=、,

23. 以下选项中可作为 C 语言合法整数的是()。

- A. 10110B B. 0386 C. 0Xffa D. x2a2

24. 下列变量定义中合法的是()。

- A. short _a=1-.le-1; B. double b=1+5e2.5;
C. long do=0xfdaL; D. float 2_and=1-e-3;

25. 与数学式子 $\frac{9x^n}{2x-1}$ 对应的 C 语言表达式是()。

- A. $9*x^n/(2*x-1)$ B. $9*x**n/(2*x-1)$
 C. $9*pow(x,n)*(1/(2*x-1))$ D. $9*pow(n,x)/(2*x-1)$
26. 若有代数式 $\frac{3ab}{cd}$, 则不正确的 C 语言表达式是()。
- A. $a/c/d*b*3$ B. $3*a*b/c/d$
 C. $3*a*b/c*d$ D. $a*b/d/c*3$
27. 已知各变量的类型说明如下:

```
int m=8,n, a, b;
unsigned long w=10;
double x=3.14, y=0.12;
```

- 则以下符合 C 语言语法的表达式是()。
- A. $a+=a--(b=2)*(a=8)$ B. $n=n*3=18$
 C. $x\%3$ D. $y=float(m)$
28. 以下符合 C 语言语法的赋值表达式是()。
- A. $a=9+b+c=d+9$ B. $a=(9+b, c=d+9)$
 C. $a=9+b, b++, c+9$ D. $a=9+b++=c+9$
29. 已知字母 A 的 ASCII 码为十进制数 65, 且 S 为字符型, 则执行语句 “S='A'+'6'-'3'”; 后, S 中的值为()。
- A. 'D' B. 68 C. 不确定的值 D. 'C'
30. 在 C 语言中, 要求运算数必须是整型的运算符是()。
- A. / B. ++ C. *= D. %
31. 若有说明语句 “char s='\72';”, 则变量 s()。
- A. 包含一个字符 B. 包含两个字符
 C. 包含三个字符 D. 说明不合法, s 的值不确定
32. 若有定义 “int m=7; float x=2.5, y=4.7;”, 则表达式 $x+m\%3*(int)(x+y)\%2/4$ 的值是()。
- A. 2.500000 B. 2.750000 C. 3.500000 D. 0.000000
33. 在 C 语言中, char 型数据在内存中的存储形式是()。
- A. 补码 B. 反码 C. 原码 D. ASCII 码
34. 设变量 x 为 float 类型, m 为 int 类型, 则以下能实现将 x 中的数值保留小数点后两位, 第三位进行四舍五入运算的表达式是()。
- A. $x=(x*100+0.5)/100.0$ B. $m=x*100+0.5, x=m/100.0$
 C. $x=x*100+0.5/100.0$ D. $x=(x/100+0.5)*100.0$
35. 表达式 $13/3*\sqrt{16.0}/8$ 的数据类型是()。
- A. int B. float C. double D. 不确定
36. 设以下变量均为 int 类型, 则值不等于 7 的表达式是()。
- A. $(m=n=6, m+n, m+1)$ B. $(m=n=6, m+n, n+1)$
 C. $(m=6, m+1, n=6, m+n)$ D. $(m=6, m+1, n=m, n+1)$
37. 假设所有变量均为整型, 则表达式 $(x=2, y=5, y++, x+y)$ 的值是()。

A. 7 B. 8 C. 6 D. 2

38. 已知 s 是字符型变量, 下面不正确的赋值语句是()。

A. s='\012'; B. s='u+v'; C. s='1'+2'; D. s=1+2;

39. 已知 s 是字符型变量, 下面正确的赋值语句是()。

A. s='abc'; B. s='\08'; C. s='\xde'; D. s="\";

40. 若有以下定义, 则正确的赋值语句是()。

```
int x,y;
float z;
```

A. x=1,y=2; B. x=y=100 C. x++; D. x=int (z);

41. 设 x、y 均为 float 型变量, 则不正确的赋值语句是()。

A. ++x; B. x*=y-2; C. y=(x%3)/10; D. x=y=0;

42. 下列语句中符合 C 语言的赋值语句是()。

A. a=7+b+c=a+7; B. a=7+b++=a+7;
C. a=7+b,b++,a+7 D. a=7+b,c=a+7;

43. 设有 “int a=1,b=2,c=3,d=4,m=2,n=2;”, 执行 (m=a>b)&&(n=c>d) 后 n 的值为()。

A. 1 B. 2 C. 3 D. 4

44. 设整型变量 a 为 5, 使 b 不为 2 的表达式是()。

A. b=a/2 B. b=6-(--a)
C. b=a%2 D. b=a>3?2:1

45. 假定有变量定义 “int k=7,x=12;”, 则能使值为 3 的表达式是()。

A. x%=k%=5 B. x%=k-k%5
C. x%=k-k%3 D. (x%=k)-(k%=5)

46. 已有声明 “int x,m=1,n=0;”, 则执行赋值语句 “x=m<n++?m++:n++;” 后, 变量 x、a、b 的值分别为()。

A. 0 1 0 B. 1 2 1 C. 1 1 2 D. 1 1 1

47. 设有语句 “char s[]=”\101”;”, 则 sizeof(s) 的值是()。

A. 0 B. 1 C. 2 D. 3

48. 若 x=0, y=3, z=3, 以下表达式值为 0 的是()。

A. !x B. x<y? 1:0 C. x%2&& y==z D. y=x||z/3

二、填空题

1. C 语言的标识符必须以_____开头。

2. 字符常量是由_____括起来的字符。

3. C 语言中 “转义字符” 是以符号_____开头的。

4. 字符变量中存放的是字符对应的_____编码值。

5. 在定义变量的同时给变量赋予初值, 称为变量的_____。

6. 字符串 “ab\045\\x66” 的长度是_____。

7. 代数表达式 $\sqrt{(\sin 60^\circ + 1) \frac{\sin 30^\circ + 1}{\cos x}}$ 的 C 语言算术表达式是_____。

8. 代数表达式 $|1-x^{3.6}|$ 的 C 语言算术表达式是_____。
9. 代数表达式 $\text{ctgx} + \log_2 7$ 的 C 语言算术表达式是_____。
10. 代数表达式 $\frac{e^x + e^{-x}}{2}$ 的 C 语言算术表达式是_____。

三、简答题

1. 下列哪些是 C 语言中的合法常量？

01 018 195 0xhh 0xff21 '\ff' e5 -0.e5 1.0e0.5
"123" '\ffl'

2. 下列哪些是 C 语言中的合法标识符？

9xy _year _123 ABC π e pi int int_ file.c

3. 简述 C 语言中字符与字符串的区别。
4. 在 C 语言中为什么变量要“先定义，后使用”？
5. 简述 C 语言中各种类型数据间运算的类型转换的规则。
6. 简述本章中学到的各种数据类型及其长度、格式、取值范围。

四、上机操作题

1. 设长方形的高为 1.5，宽为 2.3，编程求该长方形的周长和面积。
2. 编写一个程序，将大写字母 A 转换为小写字母 a。
3. 指出下面程序的错误，并改正。

```
main()
{
    int  a=2;b=3;

    c+=a+b
    scanf("%d,%d,%d",a,b,c);
    printf("a=%d,b=%d,c=%d",a,b,c);
}
```

4. 写出下面程序的输出结果。

```
#include<stdio.h>
int main()
{
    int a=3,b=4,c=1,max,t;
    if(a>b,a>c)
        max=a;
    else
        max=0;
    t=(a+3,b+1,++c);
    printf("max=%d,t=%d\n",max,t);
    return 0;
}
```

第4章 数据的输入/输出

数据的输入/输出是程序设计中用最普遍的基本操作。本章主要介绍输入/输出的概念、实现方法，以及基本类型数据输入/输出函数。

学习目标

- 了解 C 语言实现输入/输出的方法
- 掌握字符输入/输出函数的使用方法
- 掌握格式化输入库函数 `scanf()` 的用法
- 掌握格式化输出库函数 `printf()` 的用法

4.1 输入/输出概述

输入/输出(简称 I/O)是程序的基本组成部分。运行所需的数据通常要从外部设备(如键盘、文件等)输入，程序的运行结果通常也要输出到外部设备(如显示器、打印机、文件等)。

C 语言中没有提供专门的输入/输出语句，输入/输出操作是通过调用 C 语言的标准函数库来实现的。C 语言的标准函数库中提供许多用于标准输入/输出操作的库函数，使用这些标准输入/输出函数时，只要在程序的开始位置加上如下一行编译预处理命令即可：

```
#include<stdio.h>
```

它的作用是：将输入/输出函数的头文件 `stdio.h` 包含到用户源文件中。其中，`h` 为 `head` 之意，`std` 为 `standard` 之意，`i` 为 `input` 之意，`o` 为 `output` 之意。

本章只涉及 ANSI C 标准定义的输入/输出函数。由于各种机器间的差异较大，因此 ANSI C 没有定义实现各种屏幕控制(如鼠标定位)或图形显示的函数，多数 C 编译程序都在标准 C 的基础上增加了自己的显示和图形库函数，当然，这些函数只能运行于特定的编译环境。

4.2 非格式化字符的输入/输出

`getchar()` 和 `putchar()` 是专门用于字符输入/输出的函数。其中，`getchar()` 用于从键盘读入一个字符，它等待击键，待用户击键后，将读入值返回，并自动将用户击键结果回显到屏幕上；`putchar()` 则把字符写到屏幕的当前光标位置。这两个函数的使用格式如下：

```
变量 = getchar();  
putchar(变量);
```

【案例 4-1】 函数 `getchar()` 和 `putchar()` 的使用。

```
#include<stdio.h>  
int main()
```



```

{
    char ch;
    printf("Press a key and then press Enter:");
    ch=getchar();          /*从键盘输入一个字符，并将该字符存入变量 ch*/
    printf("You pressed:");
    putchar(ch);           /*在屏幕上显示变量 ch 中的字符*/
    putchar('\n');         /*输出一个回车换行符*/
    return 0;
}

```

运行结果如图 4-1 所示。

```

Press a key and then press Enter:A
You pressed: A

-----
Process exited with return value 0
Press any key to continue . . .

```

图 4-1 运行结果

从案例 4-1 可以得出以下结论。

- ① 函数 `putchar()` 的作用是向终端显示器屏幕输出一个字符，这个字符可以是可打印字符，也可以是转义序列，函数 `putchar()` 的参数就是待输出的字符。
- ② 函数 `getchar()` 的作用是从系统隐含指定的输入设备(即终端键盘)输入一个字符，按回车键表示输入结束。函数 `getchar()` 没有参数，函数的返回值就是从终端键盘读入的字符。

4.3 格式化数据的输出

`stdio` 库提供的格式化输出函数 `printf()` 的主要功能是，向显示器(标准输出设备)输出指定的若干个格式化的数据，适用于任意的数据类型。函数的一般调用格式为：

```
printf("格式控制字符串", 输出列表);
```

例如：

```
printf("i=%d,f=%f,c=%c\n",i,f,c);
```

└──────────┘
└──┘
 格式控制字符串 输出列表

格式控制字符串(Format Control String)是用双引号括起来的字符串，也简称为格式字符串(Format String)，输出值参数表中可有多多个输出值，也可没有(当只输出一个字符串时)。格式控制字符串包括两部分，分别为：如表 4-1 所示的格式转换说明符(Conversion Specifiers)和需原样输出的普通的文本字符(Literal Characters)。

格式控制字符串描述了输出格式。在一个格式控制字符串中可以有多个转换说明。输出值参数表逐个对应格式控制字符串中的每个转换说明。每个格式转换说明(Conversion Specification)都以一个%开始，以一个格式字符作为结束，用于指定各输出值参数的输出格式。

表 4-1 格式转换说明符

格式转换说明符	用 法
%d 或%i	输出带符号的十进制整数，正数的符号省略
%u	以无符号的十进制整数形式输出
%o	以无符号的八进制整数形式输出，不输出前导符 0
%x (%#x)	以无符号十六进制整数形式(小写)输出，不输出前导符 0x
%X (%#X)	以无符号十六进制整数形式(大写)输出，不输出前导符 0x
%c	输出一个字符
%s	输出字符串
%f	以十进制小数形式输出实数(包括单、双精度)，整数部分全部输出，隐含输出 6 位小数，输出的数字并非全部是有效数字，单精度实数的有效位数一般为 7 位，双精度实数的有效位数一般为 16 位
%e	以指数形式(小写 e 表示指数部分)输出实数，要求小数点前必须有且仅有 1 位非零数字
%E	以指数形式(大写 E 表示指数部分)输出实数
%g	根据数据的绝对值大小，自动选取 f 或 e 格式中输出宽度较小的一种使用，且不输出无意义的 0
%G	根据数据的绝对值大小，自动选取 f 或 E 格式中输出宽度较小的一种使用，且不输出无意义的 0
%p	以主机的格式显示指针，即变量的地址
%n	令 printf() 把自己到%n 位置已经输出的字符总数放到后面相应的输出项指向的整型变量中，printf() 函数返回后，%n 对应的输出项指向的变量中存放的整型值为出现%n 时已经由 printf() 函数输出的字符总数，%n 对应的输出项是记录该字符总数的整型变量的地址
%%	显示百分号(%)

输出列表是需要输出的数据项的列表，这些输出数据项可以是变量或表达式，之间用逗号分隔。输出列表的数据类型应与格式转换说明符相匹配。每个格式转换说明符和输出列表中的输出值参数是一一对应的，如果没有输出值参数，那么格式控制字符串中就不再需要格式转换说明符了。

【案例 4-2】 格式转换说明符在 printf() 中的使用。

```
#include<stdio.h>
int main()
{
    char ch='A';
    int a=88;
    float b=23.4;
    printf("%c,%d,%d,%o,%x,%f",ch,ch,a,a,a,b);
    return 0;
}
```

程序运行结果如图 4-2 所示。

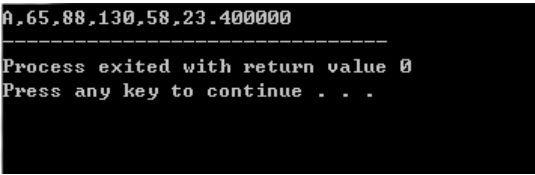


图 4-2 运行结果

在使用 printf() 函数时，应注意以下几个问题。

① 若要使用 `printf()` 函数原样输出字符串, 则无须格式控制符, 调用格式为 “`printf("字符串");`”。

② 若要利用 `printf()` 函数来显示 % 字符, 则应在格式控制字符串中连写两个 %, 例如: “`printf("%d%%", 100/4);`” 将显示 25%。

③ 对格式说明符 %c、%d、%s、%f 等可以指定输出字段的宽度。

%md: m 为指定的输出字段的宽度。如果数据的位数大于 m, 则按实际的位数输出, 否则输出时向右对齐, 左端补 “空格” 符。

%-md: m 为指定的输出字段的宽度。如果数据的位数大于 m, 则按实际的位数输出, 否则输出时向左对齐, 右端补 “空格” 符。

%mc: m 为指定的输出字段的宽度。如果 m 大于一个字符的宽度, 则输出时向右对齐, 左端补 “空格” 符。

%-mc: m 为指定的输出字段的宽度。如果 m 大于一个字符的宽度, 则输出时向左对齐, 右端补 “空格” 符。

%ms: m 为输出时字符串所占的列数。如果字符串的长度(字符个数)大于 m, 则按字符串的本身长度输出, 否则输出时, 字符串向右对齐, 左端补 “空格” 符。

%-ms: m 为输出时字符串所占的列数。如果字符串的长度小于 m, 则输出时字符串向左对齐, 右端补 “空格” 符。

也可以使用 **%m.ns**(输出占 m 列, 但只取字符串中左端的 n 个字符, 这 n 个字符输出在 m 列的右侧, 左端补 “空格” 符) 和 **%-m.ns**(同 %m.ns, 右端补 “空格” 符), 如果 $n > m$ (比如 m 不存在), 则 m 自动取 n, 即保证 n 个字符正常输出。

例如:

```
printf("%3s,%7.2s,%.4s,%-5.3s","China","China","China","China");
```

输出结果为: China,Ch,Chin,Chi (代表 “空格” 符)

%m.nf: m 为浮点型数据所占的总列数(包括小数点), n 为小数点后面的位数。如果数据的长度小于 m, 则输出时向右对齐, 左端补 “空格” 符。如果在显示时只需要对小数点后面的位数进行限制, 则 m 可以省略。如果不为 %f 指定输出的宽度, 则按系统规定的宽度输出, 此时证书部分将全部输出, 同时输出 6 位小数。

%-m.nf: m、n 的意义同上。如果数据的长度小于 m, 则输出时向左对齐, 右端补 “空格” 符。

一般来讲, 除非特殊要求, 否则在显示数据时, 可以不指定输出字段的宽度, 直接利用系统隐含的输出宽度。

④ 除格式说明符及其输出字段的宽度外, 在格式控制字符串中的其他字符将按原样输出。

4.4 格式化数据的输入

`stdio` 库提供的 `scanf()` 函数是具有格式控制的输入函数, 它可以用来输入 C 语言中任何类型的数据, 而且可以同时输入多个同类型的或不同类型的数据, 其一般调用形式如下:

`scanf("格式控制字符串", 参数地址列表);`

“格式控制字符串”的含义同 `printf()` 函数。

“参数地址列表”是由若干个地址组成的列表，用于接收输入的数据，可以是变量的地址，或字符串的首地址。

地址是由地址运算符“&”后接变量名组成的，如 `&a` 和 `&b` 分别表示变量 `a` 和变量 `b` 的地址。`&`是一个取地址运算符，`&a` 是一个表达式，其功能是求变量的地址。变量的地址是 C 编译系统分配的，用户不必关心具体的地址是多少。

`scanf()` 函数与 `printf()` 函数中的格式转换说明符基本相同，用法也基本相同，如表 4-2 所示。

表 4-2 函数 `scanf()` 的格式转换说明符

格式转换说明符	用 法
<code>%d</code> 或 <code>%i</code>	输入十进制整数
<code>%o</code>	输入八进制整数
<code>%x</code>	输入十六进制整数
<code>%c</code>	输入一个字符，空白字符(包括空格、回车、制表符)也作为有效字符输入
<code>%s</code>	输入字符串，遇到第一个空白字符(包括空格、回车、制表符)时结束
<code>%f</code> 或 <code>%e</code>	输入实数，以小数或指数形式输入均可
<code>%%</code>	输入一个百分号(%)

58

参数地址列表是由若干变量的地址组成的列表，这些参数之间用逗号分隔。函数 `scanf()` 要求必须指定用来接收数据的变量的地址，每个转换说明符都对应一个存储数据的目标地址。如果没有指定存储数据的目标地址，虽然编译器不会提示出错，但会导致数据无法正确读入指定的内存单元。

在函数 `scanf()` 的 `%` 与格式字符之间也可插入格式修饰符，如表 4-3 所示。

表 4-3 函数 `scanf()` 的格式修饰符

格式修饰符	用 法
英文字母 <code>l</code>	加在格式字符 <code>d</code> 、 <code>i</code> 、 <code>o</code> 、 <code>x</code> 、 <code>u</code> 之前，用于输入 <code>long</code> 型数据 加在格式字符 <code>f</code> 、 <code>e</code> 之前，用于输入 <code>double</code> 型数据
英文字母 <code>L</code>	加在格式字符 <code>f</code> 、 <code>e</code> 之前，用于输入 <code>long double</code> 型数据
英文字母 <code>h</code>	加在格式字符 <code>d</code> 、 <code>i</code> 、 <code>o</code> 、 <code>x</code> 之前，用于输入 <code>short</code> 型数据
域宽 <code>m</code> (正整数)	指定输入数据的宽度(列数)，系统自动按此宽度截取所需数据
忽略输入修饰符*	表示对应的输入项在读入后不赋给相应的变量，即让 <code>scanf()</code> 函数从输入流中读取任意类型的数据并将其丢弃，而不是将其赋值给一个变量，因此也称为赋值抑制字符(Assignment Suppression Character)

值得注意的是，在输入函数 `scanf()` 的格式控制字符串的格式转换说明符中指定显示精度是错误的，只有在输出函数 `printf()` 的格式转换说明符中才能够指定显示精度。

【案例 4-3】 求任意两个整数之和。

```
#include<stdio.h>
int main()
{
    int a,b;
    printf("Please input a and b:");
    scanf("%d%d",&a,&b);
}
```

```
printf("a+b=%d\n",a+b);
return 0;
}
```

若输入 8、9，则程序的运行结果如图 4-3 所示。

```
Please input a and b: 8 9
a+b=17

-----
Process exited after 8.413 seconds with return value 0
请按任意键继续. . .
```

图 4-3 运行结果

4.5 本章小结

本章介绍了 C 语言常用标准输入/输出函数的使用方法。函数 `getchar()` 和 `putchar()` 用于字符输入/输出操作，函数 `scanf()` 和 `printf()` 用于格式输入/输出操作，可以控制按各种格式进行任意类型数据的读/写操作。`scanf()` 和 `printf()` 的用法较为灵活，需要读者在实际应用中领会。

4.6 习题

一、单选题

- 语句 “`intf("##\b\b##");`” 在屏幕上的输出结果是()。
 - `##bb##`
 - `##\b\b##`
 - `####`
 - `##`
- 在下面的语句中，错误的赋值语句是()。
 - `x=(y=(z=2,m=3));`
 - `x=i+++3;`
 - `x-x/y=2`
 - `x=y==x+1`
- 语句 “`intf("%d\n",sizeof("\t"\064\x0D\n));`” 的输出结果是()。
 - 16
 - 7
 - 6
 - 5
- 有定义 “`t x=-2;`” 当执行语句 “`printf("%d,%u",x,x);`” 后输出()。
 - 2,-2
 - 2,65534
 - 65534,-2
 - 65534,65534
- 使用格式化输入函数 `scanf()` 输入一个无符号数给变量 `x`，以下不正确的是()。
 - `scanf("%u",&x)`
 - `scanf("%x",&x)`
 - `scanf("%o",&x)`
 - `scanf("%d",&x)`
- 已知 “`t a=1,b=-1;`”，语句 “`printf("%d\n",a--,++b);`” 的输出结果是()。
 - 1
 - 0
 - 1
 - 语句错误

7. 以下程序的输出结果是()。

```
main ()
{
    int a=12,b=12;
    printf("%d %d\n",--a,++b);
}
```

- A. 10 10 B. 12 12 C. 11 10 D. 11 13
8. 执行语句 “printf("|%10.5f| \n",12345.678);” 的输出是()。
- A. |12345.67800| B. |12345.6780|
C. |12345.67800| D. |12345.678|
9. 若有定义 “int a=10;”, 执行 “printf("%d",-a++);” 语句后变量 a 的值是()。
- A. 10 B. -10 C. 9 D. -9
10. putchar() 函数可以向终端输出一个()。
- A. 整型变量表达式值 B. 字符串
C. 实型变量值 D. 字符或字符型变量值
11. 以下程序段的输出结果是()。

```
int a=12345;
printf("%2d\n",a);
```

- A. 12 B. 34
C. 12345 D. 提示出错、无结果
12. 若 x 和 y 均定义为 int 型, z 定义为 double 型, 以下不合法的 scanf() 函数调用语句为()。
- A. scanf("%d%lx,%le",&x,&y,&z); B. scanf ("%2d*%d%lf",&x,&y,&z);
C. scanf ("%x%*d%o",&x,&y); D. scanf ("%x%o%6.2f",&x,&y,&z);
13. 有如下程序段:

```
int x1,x2;
char y1,y2;
scanf("%d%c%d%c",&x1,&y1,&x2,&y2);
```

若要求 x1、x2、y1、y2 的值分别为 10、20、A、B, 正确的数据输入是() (注: □ 代表空格)。

- A. 10A□20B B. 10□A20B
C. 10□A□20□B D. 10A20□B
14. 若变量已正确说明为 float 类型, 要通过语句 “scanf("%f%f%f",&a,&b,&c);” 给 a 赋予 10.0, b 赋予 22.0, c 赋予 33.0, 不正确的输入形式为()。
- A. 10<回车> B. 10.0,22.0,33.0<回车>
22<回车>
33
C. 10.0<回车> D. 10 22<回车>
22.0 3.0<回车> 33<回车>

15. 有如下程序段, 若要求 x1、x2、y1、y2 的值分别为 10、20、A、B, 正确的数据输入是() (注: □ 代表空格)。

```
int x1,x2;
char y1,y2;
scanf("%d%d",&x1,&x2);
scanf("%c%c",&y1,&y2);
```

- A. 1020AB B. 10□20□ABC
C. 10□20 D. 10□20AB

AB

16. 有定义 “int a=-2;” 和输出语句 “printf(“%8lx”,a);”, 以下正确的叙述是()。

- A. 整型变量的输出格式转换说明符只有%d 一种
B. %x 是格式转换说明符的一种, 它适用于任何一种类型的数据
C. %x 是格式转换说明符
D. %8lx 不是错误的格式转换说明符, 其中数字 8 规定了输出字段的宽度

17. 有如下程序段, 对应正确的数据输入是()。

```
float x,y;
scanf("%f%f",&x,&y);
printf("a=%f,b=%f",x,y);
```

- A. 2.04<回车> B. 2.04,5.67<回车>
5.67<回车>
C. A=2.04,B=5.67<回车> D. 2.055.67<回车>

18. 有如下程序段, 从键盘输入数据的正确形式应是() (注: □ 代表空格)。

```
float x,y,z;
scanf("x=%d,y=%d,z=%d",&a,&y,&z);
```

- A. 123 B. x=1,y=2,z=3
C. 1,2,3 D. x=1□y=2□z=3

19. 以下说法正确的是()。

- A. 输入项可以为一个实型常量, 如 “scanf(“%f”,3.5);”
B. 只有格式控制, 没有输入项, 也能进行正确输入, 如 “scanf(“a=%d,b=5d”);”
C. 当输入一个实型数据时, 格式控制部分应规定小数点后的位数, 如 “scanf(“%4.2f”,&f);”
D. 当输入数据时, 必须指明变量的地址, 如 “scanf(“%f”,&f);”

20. 根据定义和数据的输入方式, 输入语句的正确形式为() (注: □ 代表空格)。

已有定义: float x,y;

数据的输入方式: 1.23<回车>

4.5<回车>

- A. scan(“%f%f”,&x,&y); B. scanf(“%f%f”,&x,&y);
C. scanf(“%3.2f□%2.1f”,&x,&y); D. scanf(“%3.2f%2.1f”,&x,&y);

21. 根据下面的程序及数据的输入和输出形式, 程序中输入语句的正确形式应该为 ()。

```
#include "stdio.h"
main()
{   char s1,s2,s3;
    输入语句;
    printf("%c%c%c",s1,s2,s3);
}
输入形式: A B C<回车>           (注: 代表空格)
输出形式: A B
```

- A. scanf("%c%c%c",&s1,&s2,&s3);
 B. scanf("%c %c %c",&s1, &s2,&s3);
 C. scanf("%c,%c,%c",&s1,&s2,&s3);
 D. scanf("%c%c",&s1, &s2,&s3);
22. 以下程序的执行结果是 ()。

```
#include "stdio.h"
main()
{   int x=2,y=3;
    printf("x=%d,y=%d\n",x,y);
}
```

- A. x=%2,y=%3
 B. x=%d,y=%d
 C. x=2,y=3
 D. x=%d,y=%d

23. 以下程序的输出结果是 () (注: 代表空格)。

```
#include "stdio.h"
main()
{   printf("\nstring1=%15s","programming");
    printf("\nstring2=%-5s","boy");
    printf("string3=%2s","girl");
}
```

- A. string1=programming * B. string1=programming*
 string2=boy* string2=boy *string3=gi*
 string3=gi*
 C. string1=programming * D. string1=programming*
 string2= boy*string3=girl* string2=boy *string3=girl*

24. 根据题目中已给出的数据的输入和输出形式, 程序中输入和输出语句的正确内容是 ()。

```
#include "stdio.h"
main()
{   int a;
    float b;
    输入语句
```


输出语句

}

输入形式: 1 2.3<回车> (注: 代表空格)

输出形式: a+b=3.300

- A. scanf("%d%f",&a,&b); B. scanf("%d%3.1f",&a,&b);
 printf("\na+b=%5.3f",a+b); printf("\na+b=%f",a+b);
 C. scanf("%d,%f",&a,&b); D. scanf("%d%f",&a,&b);
 printf("\na+b=%5.3f",a+b) printf("\na+b=%f",a+b);

25. 阅读以下程序, 当输入数据的形式为: 12,34, 正确的输出结果为()。

```
#include "stdio.h"
main()
{   int a,b;
    scanf("%d%d",&a,&b);
    printf("a+b=%d\n",a+b);
}
```

- A. a+b=46 B. 有语法错误 C. a+b=12 D. 不确定值

26. 若有定义 “int x,y; char s1,s2,s3;”, 并有以下输出数据(注: 代表空格):

```
1 2<回车>
U V W<回车>
```

则能给 x 赋整数 1, 给 y 赋整数 2, 给 s1 赋字符 U, 给 s2 赋字符 V, 给 s3 赋字符 W 的正确程序段是()。

- A. scanf("x=%dy=%d",&x,&y);s1=getchar();s2=getchar();s3=getchar();
 B. scanf("%d%d",&x,&y);s1=getchar();s2=getchar();s3=getchar();
 C. scanf("%d%d%c%c%c",&x,&y,&s1,&s2,&s3);
 D. scanf("%d%d%c%c%c%c%c",&x,&y,&s1,&s1,&s2,&s2,&s3,&s3);

二、填空题

1. 以下程序的执行结果是_____。

```
#include "stdio.h"
main()
{   short i=-1,j=1;
    printf("dec:%d,oct:%o,hex:%x,unsigned:%u\n",i,i,i,i);
    printf("dec:%d,oct:%o,hex:%x,unsigned:%u\n",j,j,j,j);
}
```

2. 以下程序的执行结果是_____。

```
#include "stdio.h"
main()
{   char s='b';
    printf("dec:%d,oct:%o,hex:%x,ASCII:%c\n",s,s,s,s);
}
```

3. 以下程序的执行结果是_____。

```
#include "stdio.h"
main()
{   float pi=3.1415927;
    printf("%f,%.4f,%.4f,%10.3f",pi,pi,pi,pi);
    printf("\ne,%.4e,%.4e,%10.3e",pi,pi,pi,pi);
}
```

4. 以下程序的执行结果是_____。

```
#include "stdio.h"
main()
{   char c='c'+5;
    printf("c=%c\n",c);
}
```

5. 以下程序输入 1□2□3 后的执行结果是_____ (注：□代表空格)。

```
#include "stdio.h"
main()
{   int i,j;
    char k;
    scanf("%d%c%d",&i,&k,&j);
    printf("i=%d,k=%c,j=%d\n",i,k,j);
}
```

6. 有以下程序，输入 9876543210 后的执行结果是_____；输入 98□76□543210 后的执行结果是_____；输入 987654□3210 后的执行结果是_____ (注：□代表空格)。

```
#include "stdio.h"
main()
{   int x1,x2;
    char y1,y2;
    scanf("%2d%3d%3c%c",&x1,&x2,&y1,&y2);
    printf("x1=%d,x2=%d,y1=%c,y2=%c\n",x,y);
}
```

7. 若 x 和 y 均为 int 型变量，则以下语句的功能是_____。

```
x+=y;
y=x-y;
x-=y;
```

8. 有一个输入函数语句“scanf("%d",k);”，则不能使 float 类型变量 k 得到正确数值的原因是_____。

9. 有如下程序段，输入数据 12345f1678 后，u 的值是_____，v 的值是_____。

```
int u;
float v;
scanf("%3d%f",&u,&v);
```

三、简答题

1. 分析下面程序的输出。

```
#include<stdio.h>
void main()
{
    short j=50000;
    unsigned short uj=50000;
    printf("%hd,%hu\n",j,uj);
}
```

2. 负数可以赋值给不能取负值的无符号整型变量吗？分析下面程序的输出。

```
#include<stdio.h>
void main()
{
    unsigned short j=-1;
    printf("%hu\n",j);
}
```

3. 分析下面的程序。

```
#include<stdio.h>
void main()
{
    short i,j,k;
    scanf("%hd%hd%hd",&i,&j,&k);
    printf("%hd,%hd,%hd\n",i,j,k);
}
```

(1) 用户的输入为 23 - 023 - 0x23 ✓，写出程序的输出结果。

(2) 将语句 “scanf(“%hd%hd%hd”,&i,&j,&k);” 改为 “scanf(“%hd%ho%hx”,&i,&j,&k);”，用户的输入数据不变，再次写出程序的输出结果。

4. 改正下面程序中的错误。

```
#include<stdio.h>
void main()
{
    double lfd;
    scanf("%e",&lfd);
    printf("%e\n",lfd);
}
```

5. 把用户输入的一个整数和一个字符分别存入 short 型变量 j 和字符型变量 ca 时，语句 “scanf(“%c%d”,&ca,&j);” 和语句 “scanf(“%d%c”,&j,&ca);” 有无区别？

第5章 结构化程序设计

前几章讲解的是 C 语言的基本语法知识,仅靠这些还无法编写出一个完整的 C 语言程序。C 语言程序的编写还需要加入业务逻辑,对流程进行控制。在教师工资管理系统中任何一个功能都有一定的操作步骤,有从上往下依次执行的,有根据某个输入的值来确定执行某个功能的,也有某个操作需要重复执行多次的,这就需要对操作步骤进行控制。本章将进一步围绕程序的三种基本结构和程序控制语句来讲解一些简单的程序设计问题。

学习目标

- 理解算法的概念
- 能够使用流程图画出顺序、选择、循环三种语句的执行流程
- 熟练使用 if、switch 语句判断各种选择情况,掌握嵌套语句的使用
- 熟练使用 while、do...while、for 三种循环结构语句,解决实际问题
- 熟练掌握 break、continue、goto 语句的用法

5.1 算法

算法是用来解决问题的,问题的解决方案就是算法。在程序设计过程中算法就是对数据的处理过程。

5.1.1 算法的概念

算法(Algorithm)是指解题方案准确而完整的描述。例如,求解一元二次方程,就有很多解决方案,如公式法、配方法、直接开平方法、因式分解法等。新学期开学,新生报到,从家到学校的交通方式选择问题,也有很多解决方案,如乘坐动车、乘坐火车、乘坐飞机等,在本市的可能会自己开车或乘坐公交车,离学校近的可能会选择步行。每一种方案都能解决从家到学校的问题,因此每一种方案都是一种算法。

同样的,在计算机中,算法也是对某一个问题求解方法的描述,只是它的表现形式是计算机指令的有序序列,执行这些指令就能解决特定的问题。例如,用计算机求解三个数中的最小数,就是解决一个问题,而找出最小数的一种方法就是一个算法。

一个算法,尤其是一个成熟的算法,应该具有以下 5 个重要的特征。

① 有穷性(Finiteness):在执行有限的步骤之后,自动结束,不会无限循环,并且每一个步骤在可接受的时间内完成。这里的有穷概念不是数学意义上的,而是指在实际应用中可以接受的、合理的时间和步骤。

② 确定性(Definiteness):算法的每一个步骤都具有确定的含义,不会出现二义性,即在相同条件下,只有一个执行路径,相同的输入,将有相同的输出结果。

③ 可行性 (Effectiveness): 算法中执行的任何计算都是可以被分解为基本步骤的可执行操作步骤, 即每个计算步骤都可以在有限时间内完成 (也称为有效性)。

④ 输入项 (Input): 一个算法有零个或多个输入, 以刻画运算对象的初始情况。所谓零个输入是指, 算法本身规定了初始条件, 有的输入项需要在算法执行过程中输入。

⑤ 输出项 (Output): 一个算法有一个或多个输出, 以反映对输入数据加工后的结果。没有输出的算法是毫无意义的, 输出是算法进行信息加工后得到的结果。

解决问题的算法有多种, 每一种算法需要的资源是不同的, 人们总是追求使用最少的资源来解决问题, 这就是算法的复杂度。

算法复杂度是指算法在编写成可执行程序后, 运行时所需要的资源。资源包括时间资源和内存资源。算法分析的目的在于选择合适的算法和改进算法。一个算法的评价主要从时间复杂度和空间复杂度来考虑。

时间复杂度是指执行算法需要的计算工作量, 即对时间资源的研究。

空间复杂度是指执行算法需要的内存空间, 即对内存资源的研究。

5.1.2 结构化程序设计的三种基本结构

在程序设计中, 程序语句可以按照结构化程序设计思想构成三种基本结构, 即: 顺序结构、分支结构和循环结构。算法可以同时包括其中的一种、两种或三种。

在程序执行过程中, 程序语句大多是按照书写顺序依次执行的, 这称为顺序结构。但是, 为了能处理某些复杂问题, 程序也需要根据不同条件选择不同的程序段, 或者重复执行某些特定的程序段, 前者称为分支结构, 后者称为循环结构。

(1) 顺序结构

在顺序结构中, 程序是按照语句的书写顺序依次执行的。

(2) 分支结构

在分支结构中, 程序首先判断条件是否成立, 然后选择执行不同的程序段。

(3) 循环结构

在循环结构中, 程序根据判断条件是否成立, 来决定是否重复执行某个程序段。这样可以避免重复书写需要多次执行的语句, 从而降低程序长度。

5.1.3 流程图

在程序设计时, 应该先制定实现程序功能的算法。描述算法较为常用方法有: 伪代码、N-S 结构化流程图、PAD 图和流程图法。用得最多的是流程图法, 下面进行简单介绍。

流程图是描述问题处理步骤的一种常用图形工具, 它由一些图框和流程线组成。使用流程图描述问题的处理步骤形象直观、便于阅读。画流程图时必须按照功能选用相应的流程图符号, 常用的流程图符号如图 5-1 所示。

如图 5-1 所示的流程图符号中, 列举了 4 个图框、1 个流程线, 具体说明如下:

起止框用于表示流程的开始或结束;

输入/输出框用平行四边形表示, 在平行四边形内可以写明输入/输出的内容;

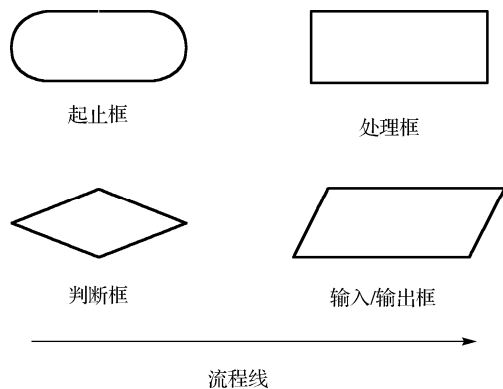


图 5-1 流程图符号

判断框用菱形表示，它的作用是对条件进行判断，根据条件是否成立来决定如何执行后续的操作；

处理框用矩形表示，它代表程序中的处理功能，如算术运算和赋值等；

流程线用实心单向箭头表示，可以连接不同位置的图框，流程线的标准流向是从左到右和从上到下的，可用流程线指示流向。

流程图的优点如下：

采用简单规范的符号，画法简单；

结构清晰，逻辑性强；

便于描述，容易理解。

顺序结构的流程图如图 5-2 所示，分支结构的流程图如图 5-3 所示，循环结构的流程图如图 5-4 所示。

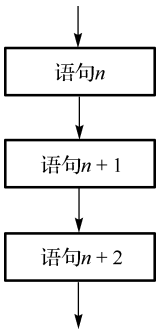


图 5-2 顺序结构

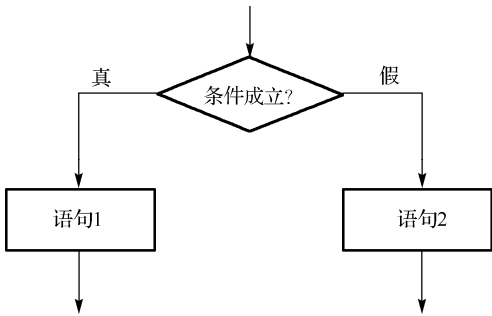


图 5-3 分支结构

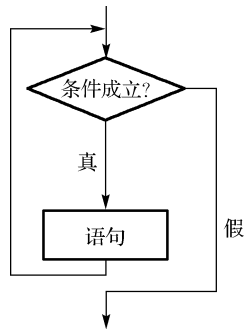


图 5-4 循环结构

【案例 5-1】 有三个数 x, y, z ，请设计一个算法找出其中最大的数，并画出算法流程图。

算法设计：

求 x, y, z 三个数中的最大数，步骤如下。

第 1 步：判断 $x > y$ 是否成立，如果成立，则进入第 2 步；如果不成立，则进入第 3 步。

第 2 步：判断 $x > z$ 是否成立，如果成立，则 x 是最大数；如果不成立，则 z 是最大数。

第 3 步：判断 $y > z$ 是否成立，如果成立，则 y 是最大数；如果不成立，则 z 是最大数。

流程图：

根据设计的算法，画出相应的流程图，如图 5-5 所示。

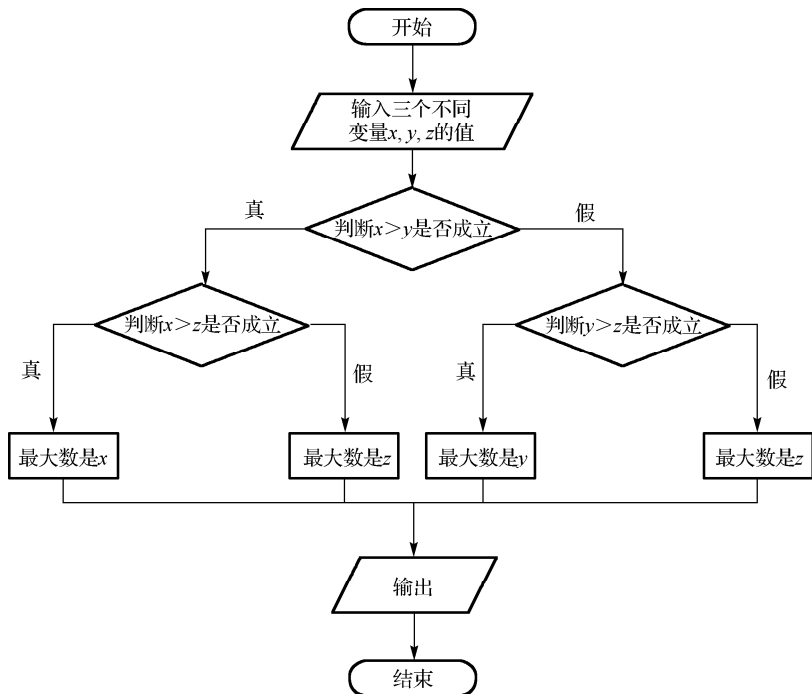


图 5-5 求三个数中的最大数

图 5-5 表示的是一个求三个数中的最大数的流程图，下面针对该流程图中的执行顺序进行说明。

第 1 步：程序开始。

第 2 步：进入输入框，输入三个变量 x, y, z 。

第 3 步：进入判断框，判断 $x > y$ 是否成立，如果成立，则进入左边的判断框，继续判断 $x > z$ 是否成立；否则进入右边的判断框，判断 $y > z$ 是否成立。

第 4 步：进入下一层判断框。如果进入的是左边的判断框，判断 $x > z$ 是否成立，如果成立，则进入下一级左边的处理框，得出最大值是 x ；如果不成立，则进入下一级右边的处理框，得出最大值是 z 。如果进入的是右边的判断框，则判断 $y > z$ 是否成立，如果成立，则进入下一级左边的处理框，得出最大值是 y ；如果不成立，则进入下一级右边的处理框，得出最大值是 z 。

第 5 步：进入输出框，输出结果。

第 6 步：进入结束框，程序运行结束。

5.2 if 分支语句

分支结构也叫选择结构，可以使某一条或几条语句在流程中不被执行或被执行。if 语句和 switch 语句就是实现选择结构的两种语句。if 语句一般用来实现量少的分支，如果分支很多的话一般采用 switch 语句。

5.2.1 if 语句中的条件表示

if 语句根据给定的条件,即表达式进行判断,表达式的值为真(非 0)或假(0),决定了 if 后紧跟的语句是否被执行。C 语言的 if 语句有三种形式,分别为:单分支 if 语句、双分支 if 语句和多分支 if 语句。另外,把基本 if 语句嵌套起来可以构成嵌套的 if 语句。

if 语句的基本形式如下:

```
if(条件表达式) 语句;
```

关键字 if 后面括号里的表达式就是用来描述条件的,该表达式通常是逻辑表达式或关系表达式,但也可以是其他表达式,如赋值表达式,甚至也可以是一个变量。

1. 关系表达式

在程序中经常需要比较两个量的大小关系,以决定程序的下一步工作。通过第 3 章的学习我们已经知道关系表达式是使用关系运算符连接起来的式子,用来表示运算对象的关系,其结果是一个逻辑值:真或假。设有定义“int a=3,b=2,c=1;”,则 a>b 的值为真,c==a 的值为假。

选择结构中,若条件表达式的值为真,则执行,否则不执行该语句。

如下为判断整数 a 能否被 2 整除的程序段:

```
int a=9;
if(a%2!=0)
    printf("%d 不能被 2 整除\n",a);
```

表达式的值为真,输出“9 不能被 2 整除”。

2. 逻辑表达式

逻辑表达式就是用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子,例如:

```
x>10||x<100    x==y&&a!=b    5&&b
```

逻辑表达式的值是一个逻辑值,即“真”或“假”。例如:

```
3>2&&8>3    结果为真
3>12||4>15   结果为假
```

它可以用来表示较为复杂的条件,如下所示,判断变量 year 是否为闰年。

是闰年的条件为:能被 4 整除,但不能被 100 整除,或者能被 400 整除。表达式为:

```
(year%4==0)&&(year%100!=0)|| (year%400==0)
```

满足条件则为闰年。

3. 任意的数值类型作表达式

如“if(7)printf("ok");”,表达式的值为 7,C 语言规定一个非 0 的数为真,则执行输出语句,输出 ok;若“if(0)printf("ok");”,则不执行输出语句。常见的还有实型、字符型、指针型数据作表达式。

注意:在 C 语言中,没有逻辑型数据类型,关系表达式或逻辑表达式的结果值只能是 1(表示结果为真)或 0(表示结果为假);在逻辑判断时,非 0 表达式可表示真,0 表达式可表示假。

5.2.2 if 语句的三种形式

if 语句的三种形式分别为：单分支 if 语句、双分支 if 语句、多分支 if 语句。

1. 单分支 if 语句

单分支 if 语句的形式如下：

```
if (表达式) 语句;
```

单分 if 语句流程图如图 5-6 所示。

首先判断表达式的值是否为真，若表达式的值非 0，则执行其后的语句；否则不执行该语句。

【案例 5-2】 求一个整数的绝对值。

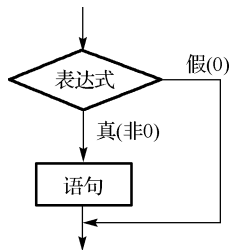


图 5-6 单分 if 语句流程图

```
#include<stdio.h>
int main()
{
    int a,b;
    printf("enter an integer: ");
    scanf("%d",&a);           /*输入整数 a*/
    b=a;
    if(a<0) b=-a;             /*求负数 a 的绝对值*/
    printf("%d 的绝对值是 %d\n",a,b); /*输出结果*/
    return 0;
}
```

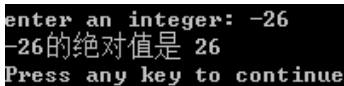


图 5-7 运行结果

程序的运行结果如图 5-7 所示。

实例分析：如果输入的整数是正数，则绝对值不变；如果是负数，则需要变成负数的相反数。

整体来看该程序是顺序结构，只不过 if 语句根据表达式的值进行了一个选择，使语句“b=-a;”要么被执行，要么不被执行。

注意：在 if 语句中，if 关键字后的表达式必须用“()”括起来，且“)”之后不加分号。

2. 双分支 if 语句

双分支 if 语句，即 if...else 语句，其一般形式为：

```
if(表达式)
    语句 1;
else
    语句 2;
```

双分支 if 语句流程图如图 5-8 所示。首先计算表达式的值，如果表达式的值为非 0，即真(True)，则执行语句 1，跳过语句 2；如果表达式的值为 0，即假(False)，则跳过语句 1，执行语句 2。然后程序继续往下执行。

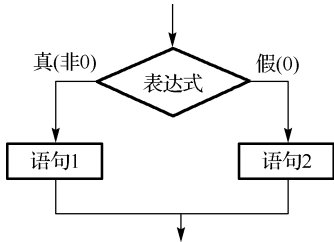
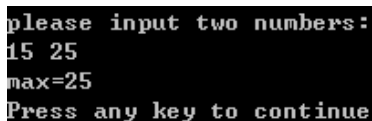


图 5-8 双分支 if 语句流程图

【案例 5-3】 输入两个整数，将大数输出。

```
#include<stdio.h>
int main()
{
    int a,b,max;
    printf("please input two numbers:\n");
    scanf("%d%d",&a,&b);
    if(a>b) max=a;        /* 如果 a 的值大于 b, 则将 a 的值赋给 max*/
    else max=b;          /* 否则将 b 的值赋给 max*/
    printf("max=%d\n",max);
    return 0;
}
```

程序的运行结果如图 5-9 所示。



```
please input two numbers:
15 25
max=25
Press any key to continue
```

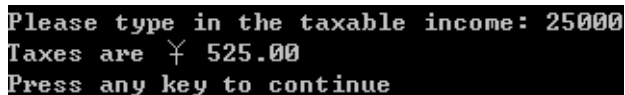
图 5-9 运行结果

72

【案例 5-4】根据收入,计算纳税金额。其中收入高于 20000 元的纳税金额分两部分,20000 以下部分按 2%收取,高于 20000 部分按 2.5%收取。

```
#include<stdio.h>
#define LOWRATE 0.02        /* 低于标准收入部分纳税率*/
#define HIGHRATE 0.025     /* 高于标准收入部分纳税率*/
#define CUTOFF 20000.0     /* 标准收入*/
int main()
{
    float taxable, taxes;
    printf("Please type in the taxable income: ");
    scanf("%f",&taxable);
    if (taxable<=CUTOFF)    /* 收入小于等于标准收入*/
        taxes=LOWRATE*taxable;
    else                    /* 收入大于标准收入*/
        taxes=HIGHRATE*(taxable-CUTOFF)+20000*0.02;
    printf("Taxes are ¥%7.2f\n",taxes);
    return 0;
}
```

程序的运行结果如图 5-10 所示。



```
Please type in the taxable income: 25000
Taxes are ¥ 525.00
Press any key to continue
```

图 5-10 运行结果

这里需要注意的是:

- ① 虽然 if 和 else 之间加了分号,但 if...else 仍是一条语句,同属于一个 if 语句;
- ② else 子句是 if 语句的一部分,和 if 语句配对使用,不能单独使用;

③ 应确保 if 和 else 子句顺序正确，这一点初学者容易犯错，往往会把本应该放在 if 后面的代码和本应该放在 else 后面的代码颠倒。

【案例 5-5】 从键盘输入三个数，求出最大数并输出。

案例分析：在案例 5-1 中的算法只是求出最大数，在本案例中，选择新的算法，先判断 $a > b$ 的结果，如果将此判断中的较大数赋值给 max 变量，再用 max 变量和 c 变量进行比较，得出三个数中的最大数，流程图如图 5-11 所示。

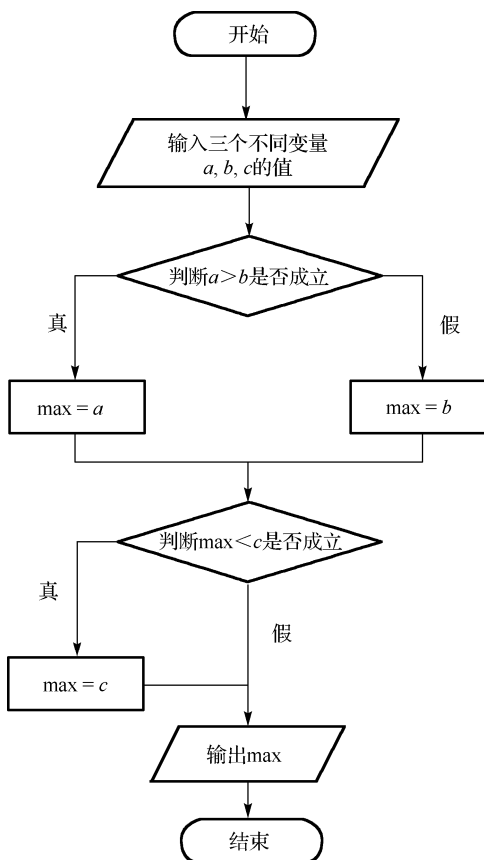


图 5-11 输出三个数中最大数的流程图

```
#include<stdio.h>
int main()
{
    float a,b,c;
    float max;
    printf("Enter three numbers\n");
    scanf("%f,%f,%f",&a,&b,&c);
    if(a>b)
        max=a;
    else
        max=b;
    if(max<c)
        max=c;
```

```
printf("Max=%f\n",max);

return 0;
}
```

程序运行结果如图 5-12 所示。

案例分析：本程序运用了一个双分支结构与一个单分支结构。将输入的三个数分别存放在变量 a, b, c 中，先利用双分支结构得到前两个数 a, b 中的大数 \max ，再用一个单分支结构将这个 \max 与第三个数 c 进行比较，将大数存放在 \max 中，这样进行比较之后得到的 \max 一定为 a, b, c 中最大的数，从而得到三个数中的最大数。与案例 5-1 的算法相比，本案例的算法更加简单明了。

```
Enter three numbers
2.4,3.5,-2.9
Max=3.500000
Press any key to continue
```

图 5-12 运行结果

3. 多分支 if 语句

多分支 if 语句，即 `if...else if...else` 形式的条件语句，一般形式为：

```
if(表达式 1) 语句 1;
else if(表达式 2) 语句 2;
...
else if(表达式 n) 语句 n;
else 语句 n+1;
```

多分支 if 语句流程图如图 5-13 所示。依次判断条件表达式的值，当出现某个值为真时，则执行其对应的语句，然后跳出整个 if 结构继续执行程序；如果所有的表达式均为假，则执行语句 $n+1$ ，然后继续执行后续程序。

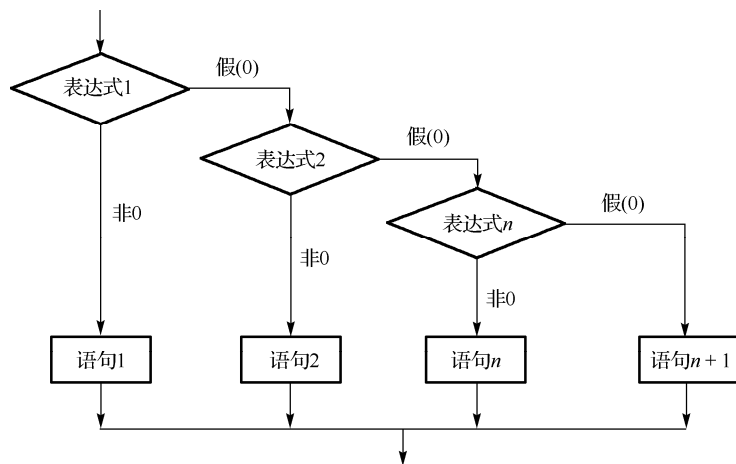


图 5-13 多分支 if 语句流程图

【案例 5-6】 实现下述分段函数，要求自变量与函数值均为双精度类型。

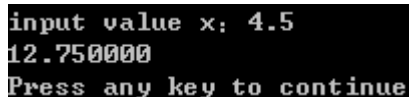
$$f(x) = \begin{cases} x \times x + 2 \times x - 5 & (x < 0 \text{ 且 } x \neq -3) \\ x \times x - 3 \times x + 6 & (0 \leq x < 20 \text{ 且 } x \neq 5 \text{ 及 } x \neq 8) \\ x \times x - 3 \times x - 10 & (x \text{ 为其他值}) \end{cases}$$

案例分析：首先查看分段函数坐标，会发现 x 的三个区间互相排斥且构成 x 轴的整体，下面使用 `if...else if...else` 语句实现。

```
#include<stdio.h>
#include<math.h>
int main()
{
    double x,y;
    int k=0;
    printf("input value x: ");
    scanf("%lf",&x);
    if(x<0&&fabs(x+3)>1e-6)
        y=x*(x+2)-5;
    else if(x>=0&&x<20&&fabs(x-5)>1e-6&&fabs(x-8)>1e-6)
        y=x*(x-3)+6;
    else
        y=x*(x-3)-10;
    printf("%f\n",y);
    return 0;
}
```

输入 x 的值为 4.5，程序运行结果如图 5-14 所示。

注意：这里自变量是双精度类型，不能直接比较（如 $x!=5$ 是错误的），要转换成绝对值形式来判断，“ $\text{fabs}(x-5)>1e-6$ ”中“ $1e-6$ ”就是“10 的-6 次方”。



```
input value x: 4.5
12.750000
Press any key to continue
```

图 5-14 运行结果

5.2.3 复合语句在分支语句中的应用

条件语句在语法上仅允许每个分支中带一条语句，而实际上分支要处理的操作往往需要多条语句才能完成，这时就要把它们用“{}”括起来，构成复合语句来执行。

复合语句就是逻辑上相关的一组用“{}”括起来的语句，它被当成一条语句来执行，当条件表达式的值为真时，“{}”内的语句全部执行，否则全部不执行。可在单个语句允许使用的任意地方使用，例如，`if` 语句也可以写成如下形式：

```
if(表达式)                /*分支语句为复合语句*/
{
    语句序列;
}
```

注意：左、右花括号应换行，并与关键字 `if` 对齐，分支内的语句相对于“{”向右缩进 4 个空格。这样层次清晰，便于程序维护。

【案例 5-7】 输入任意两个整数，按从小到大的顺序排序并输出。程序如下：

```
#include<stdio.h>
int main()
{
    int m,n,t;    /*定义三个变量 m、n、t，t 为交换顺序时所用的中间变量*/
```

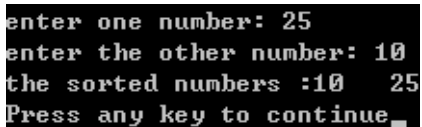
```

printf("enter one number: ");
scanf("%d",&m);
printf("enter the other number: ");
scanf("%d",&n);
if (m>n)      /*如果 m 的值大于 n，交换 m、n 的顺序*/
{
    t=m;
    m=n;
    n=t;
}
printf("the sorted numbers :%d  %d\n",m,n);
return 0;
}

```

程序运行结果如图 5-15 所示。

上述程序中，如果去掉“t=m;m=n;n=t;”外的“{}”，则运行结果如图 5-16 所示。

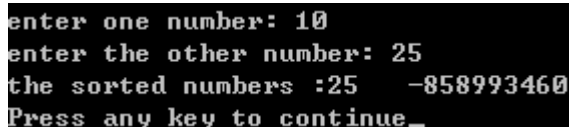


```

enter one number: 25
enter the other number: 10
the sorted numbers :10 25
Press any key to continue_

```

图 5-15 运行结果



```

enter one number: 10
enter the other number: 25
the sorted numbers :25 -858993460
Press any key to continue_

```

图 5-16 去掉“{}”后的运行结果

案例分析：这个程序应用了一个经典的交换算法，即利用中间变量。就好像交换两个杯子中的水，要用到第三个杯子，假设第三个杯子是 t，那么正确的程序为“t=m;m=n;n=t;”，此三条语句作为一个整体被执行，缺少任何一条语句都将出错，不能实现 m 与 n 的交换。所以，如果去掉了它们外面的“{}”，程序段如下：

```

...
if (m>n)
    t=m;
    m=n;
    n=t;
...

```

当 $m>n$ 时，这三条语句都将被执行，没有错误；而当 $m\leq n$ 时，只有“t=m”这一条语句不被执行，而“m=n;n=t;”这两条语句还将被执行，从而结果错误。

5.2.4 if 语句的嵌套

当有多个分支选择时，除了可以使用 if...else if...else 结构，还可以采用嵌套结构。

当 if 语句的执行语句又是 if 语句时，就构成了 if 语句的嵌套。

一般形式如下：

```

if (表达式 1)
    if (表达式 2) 语句 1;
    else 语句 2;
else

```

```
if (表达式 3) 语句 3;
else 语句 4;
```

上述结构中 if 语句中的执行语句又是 if...else 结构的, 这时将会出现多个 if 和多个 else 重叠的情况, 要特别注意 if 和 else 的配对问题。

例如以下程序段, 共 5 行:

```
① if(表达式)
②     if(表达式) 语句 1;
③ else
④     if(表达式) 语句 2;
⑤     else 语句 3;
```

在这段程序中, 有 3 个 if 子句, 2 个 else 子句。这其中的每个 else 是和哪个 if 配对的呢?

C 语言规定: 在缺少花括号的情况下, else 总是与它上面最近的, 并且没有和其他 else 配对的 if 配对。

按程序的书写格式来看, 希望第③行出现的 else 子句能和第①行出现的 if 配对, 但实际上这个例子中的第③行的 else 和第②行的 if 配对了。

那么如何才能实现第③行出现的 else 和第①行出现的 if 配对呢? 这时可以利用添加“{”的方法改变原来的配对关系。例如:

```
if (表达式)
{ if(表达式)语句 1;}
else
    if(表达式)语句 2;
    else 语句 3;
```

这样, “{”限定了内嵌 if 语句的范围, 就可以实现第③行出现的 else 和第①行出现的 if 配对。需要注意的是:

else 和 if 是成对出现的, 有 else 语句, 必定有 if 语句;

但有 if 语句, 不一定有 else 语句。

学习选择结构不要被分支嵌套迷惑, 只要掌握 else 与 if 的匹配规则, 依次匹配 if 与 else, 弄清各分支要执行的功能, 嵌套结构也就不难理解了。

总之, 嵌套的形式是千变万化的, 为了保证嵌套的层次分明和对应正确, 不要省略“{”。另外, 在书写时应尽量采取分层递进式的书写格式, 内层的语句可往右缩进几个字符(一般为 4 个), 使层次清晰, 有助于增加程序的可读性。

【案例 5-8】 求一元二次方程 $ax^2+bx+c=0$ 的解 ($a \neq 0$)。

```
#include<math.h>
#include<stdio.h>
int main()
{
    float a,b,c,deta,x1,x2,p,q;
    printf("input a,b,c:");
    scanf("%f,%f,%f",&a,&b,&c);
    deta=b*b-4*a*c;
```

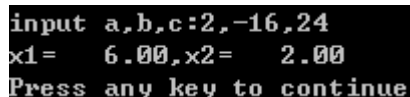
```

    if (fabs(deta)<=1e-6)                /*fabs(): 求绝对值库函数*/
        printf("x1=x2=%7.2f\n",-b/(2*a));    /*输出两个相等的实根*/
    else
    {
        if (deta>1e-6)                    /*求出两个不相等的实根*/
        {
            x1=(-b+sqrt(deta))/(2*a);
            x2=(-b-sqrt(deta))/(2*a);
            printf("x1=%7.2f,x2=%7.2f\n",x1,x2);
        }
        else                                /*求出两个共轭复根*/
        {
            p=-b/(2*a);
            q=sqrt(fabs(deta))/(2*a);
            printf("x1=%7.2f+%7.2f i\n",p,q);    /*输出两个共轭复根*/
            printf("x2=%7.2f-%7.2f i\n",p,q);
        }
    }
    return 0;
}

```

程序运行结果如图 5-17 所示。

说明：由于实数在计算机中存储时，经常会有一些微小的误差，所以本案例判断 `deta` 是否为 0 的方法是，判断 `deta` 的绝对值是否小于一个很小的数（如 10^{-6} ）。



```

input a,b,c:2,-16,24
x1= 6.00,x2= 2.00
Press any key to continue

```

图 5-17 运行结果

本案例采用了 `if` 语句的嵌套结构：先将 `deta` 分成等于零和不等于零两种情况，其中不等于零的情况中又嵌套了大于零和小于零两种情况。

5.2.5 条件运算符与条件表达式

通过第 3 章的学习我们已经知道 C 语言中的一个特殊运算符，即条件运算符（?:）。它是 C 语言中唯一的三目运算符。由条件运算符组成的条件表达式的一般形式如下：

表达式 1?表达式 2:表达式 3

其中，表达式 1 一般为关系表达式或逻辑表达式，表达式 2 和表达式 3 一般为同类型表达式。

条件表达式的求解过程是：先求解表达式 1，若表达式 1 的值不为 0，则求解表达式 2，表达式 2 的值就是条件表达式的值；若表达式 1 的值为 0，则求解表达式 3，表达式 3 的值就是条件表达式的值。

在条件语句中，若只执行单个赋值语句，常使用条件运算符来表示。这样写不但可使程序简洁，也提高了程序的运行效率。

例如，从键盘上输入一个字符，如果它是小写字母，则把它转换成大写字母输出；否则，直接输出。程序段如下：

```
printf("input a character:");
```



```
scanf("%c",&ch);
ch=(ch>='a'&&ch<='z')?(ch-32):ch;
```

如果输入大写字母 A, 则程序运行结果为 `ch=a`。

使用条件表达式时, 还应注意以下几点。

- ① 条件运算符中?和:是一对运算符, 不能分开单独使用。
- ② 条件运算符的运算优先级低于关系运算符和算术运算符, 但高于赋值运算符。因此, `max=(a>b)?a:b` 可以去掉括号, 改写为 `max=a>b?a:b`。
- ③ 条件运算符的结合方向是自右向左。例如, `a>b?a:c>d?c:d` 应理解为 `a>b?a:(c>d?c:d)`, 这也就是条件表达式嵌套的情形, 即其中的表达式 3 又是一个条件表达式。

5.3 switch 分支语句

前面介绍了由两个以上的值来控制程序流程的案例, 若将学生成绩划分为 A、B、C、D、E 等级, 利用嵌套的 if 语句或多分支 if 语句也是可以解决的, 但是如果分支太多, if 语句嵌套的层次数太多, 势必会造成程序冗长, 可读性差。

79

5.3.1 switch 语句

C 语言提供了另外一种用于多分支选择结构的 switch 语句, 它能够根据表达式的值(多于两个)来执行不同的语句。

switch 语句的一般形式如下:

```
switch(表达式)
{
    case 常量表达式 1: 语句 1;
    case 常量表达式 2: 语句 2;
    ...
    case 常量表达式 n: 语句 n;
    default          : 语句 n+1;
}
```

其执行过程是: 计算 switch 后面表达式的值, 逐个与其后的 case 常量表达式的值相比较, 当表达式的值与某个常量表达式的值相等时, 则执行其后的语句, 然后不再进行判断, 继续执行后面所有 case 后的语句。如表达式的值与所有 case 后的常量表达式的值均不相同, 则执行 default 后的语句。

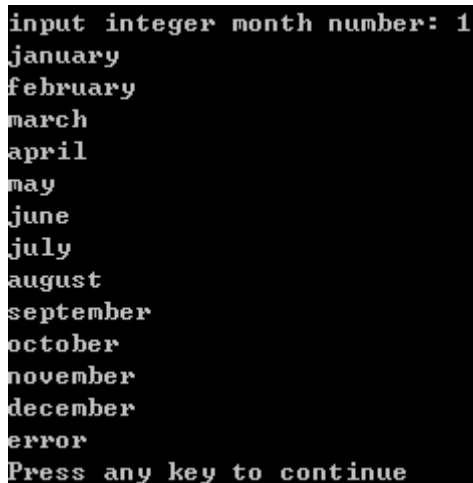
【案例 5-9】 输入一个数字月份, 输出其对应的英文单词。

```
#include<stdio.h>
int main()
{
    int a;
    printf("input integer month number: ");
    scanf("%d",&a);
```

```
switch(a)
{
    case 1:printf("january\n");
    case 2:printf("february\n");
    case 3:printf("march\n");
    case 4:printf("april\n");
    case 5:printf("may\n");
    case 6:printf("june\n");
    case 7:printf("july\n");
    case 8:printf("august\n");
    case 9:printf("september\n");
    case 10:printf("october\n");
    case 11:printf("november\n");
    case 12:printf("december\n");
    default:printf("error\n");
}
return 0;
}
```

80

输入数字 1，运行结果如图 5-18 所示。



```
input integer month number: 1
january
february
march
april
may
june
july
august
september
october
november
december
error
Press any key to continue
```

图 5-18 运行结果

这并不是我们想要的输出结果。因为在 switch 语句中，“case 常量表达式”相当于一个语句标号，表达式的值和某个标号相等，则转向该标号执行，但不能在执行完该标号的语句后跳出整个 switch 语句。为了避免上述情况，C 语言提供了 break 语句，用于跳出 switch 语句。

switch 语句一般与 break 语句配合使用，一般形式如下：

```
switch(表达式)
{
    case 常量表达式 1: 语句 1; break;
    case 常量表达式 2: 语句 2; break;
    ...
    case 常量表达式 n: 语句 n; break;
```

```

        default      : 语句  $n+1$ ;
    }

```

break 语句只能位于 **switch** 语句和循环体中，使程序立刻终止当前的 **switch** 语句，执行 **switch** 语句后面的语句。

上述 **switch** 语句的执行过程中，当表达式的值与某个常量表达式的值相等时，即执行其后的语句，然后跳出 **switch** 语句。如果表达式的值与所有 **case** 后的常量表达式的值均不相同，则执行 **default** 后的语句，然后跳出 **switch** 语句。

【案例 5-10】 输入 2016 年的一个月份，输出这个月的天数(2016 年为闰年)。

案例分析：根据输入的月份数判断天数，当月份为 1、3、5、7、8、10、12 时，天数为 31，当月份为 4、6、9、11 时，天数为 30，2016 年是闰年，所以，当月份为 2 时，天数为 29。

```

#include<stdio.h>
int main()
{
    int month,days;
    printf("input the month number: ");
    scanf("%d",&month);
    switch(month)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:days=31;break;
        case 4:
        case 6:
        case 9:
        case 11:days=30;break;
        case 2:days=29;break;
        default:days=-1;
    }
    if(days==-1)
        printf("input error! ");
    else
        printf("2016 years %d month has %d days\n",month,days);
    return 0;
}

```

输入数字月份 8，程序运行结果如图 5-19 所示。

```

input the month number: 8
2016 years 8 month has 31 days
Press any key to continue

```

图 5-19 运行结果

注意: `switch` 语句允许多情况执行相同的语句。例如, 4、6、9 和 11 月均执行 `days=30`, 可以写成:

```
case 4:case 6:case 9:case 11:days=30;
```

但不能写成:

```
case 4,6,9,11:days=30;
```

也不能写成:

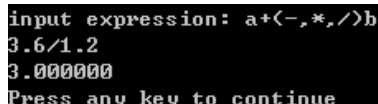
```
case 4,case 6,case 9,case 11:days=30;
```

【案例 5-11】 编写计算器程序。用户输入运算数和四则运算符, 输出计算结果。

```
#include<stdio.h>
int main()
{
    float a,b;
    char c;
    printf("input expression: a+(-,*,/)b \n");
    scanf("%f%c%f",&a,&c,&b);
    switch(c)
    {
        case '+': printf("%f\n",a+b); break;
        case '-': printf("%f\n",a-b); break;
        case '*': printf("%f\n",a*b); break;
        case '/': printf("%f\n",a/b); break;
        default: printf("input error\n");
    }
    return 0;
}
```

输入表达式 “3.6/1.2”, 程序运行结果如图 5-20 所示。

`switch` 语句用于判断字符 `c` 是+、-、*、/哪一种运算符, 然后输出相应运算值。当输入运算符不是+、-、*、/时, 给出错误提示。



```
input expression: a+(-,*,/)b
3.6/1.2
3.000000
Press any key to continue
```

图 5-20 运行结果

使用 `switch` 语句时还需注意如下问题。

- ① `switch` 关键字后的表达式可以是任何结果为整型或字符型(`long`、`int` 或 `char`)的表达式。
- ② `case` 关键字后面只能是整型或字符型的常量或常量表达式。不能处理 `case` 后为非常量的情况。例如, `if(a>1&&a<10)`, 不能使用 `switch...case` 来处理。
- ③ 在 `case` 后的各常量表达式的值不能相同, 否则会出现错误。
- ④ 在 `case` 后, 允许有多个语句, 可以不用 “{}” 括起来。
- ⑤ 每个 `case` 语句的结尾不要忘记加 `break`, 否则将导致多个分支重叠(除非有意使多个分支重叠)。
- ⑥ `default` 子句省略不用时没有语法错误。但建议使用 `default` 子句, 即使认为已经涵盖了所有情况。`default` 子句只用于检查真正的默认情况。

另外,使用 switch 语句时应注意 case 语句的排列顺序。

case 语句的排列顺序不影响输出结果,所以有很多人认为 case 语句的顺序无所谓,但事实却不是如此。如果 case 语句很少,也许可以忽略这点,但是如果 case 语句非常多,可以遵循以下规则。

- ① 按字母或数字顺序排列各条 case 语句。如果所有的 case 语句没有明显的重要差别,可按 A、B、C 或 1、2、3 等顺序排列 case 语句。这样可以很容易地找到某条 case 语句。
- ② 如果有多个正常情况和异常情况,把正常情况放在前面,把异常情况放在后面。
- ③ 按执行频率排列 case 语句。把执行频率高的情况放在前面,把不常执行的情况放在后面。经常执行的代码可能是调试时单步执行最多的代码,如果放在后面,查找起来会比较困难,放在前面,可以很快找到。

5.3.2 分支结构程序举例

【案例 5-12】 编程实现温度转换。如果输入一个华氏温度,把它转换成摄氏温度;如果输入一个摄氏温度,把它转换成华氏温度,摄氏温度转换为华氏温度的公式为: $F = C \times 9/5 + 32$ 。

```
#include<stdio.h>
int main()
{
    char tempType;
    float temp,fahren,celsius;

    printf("Enter the temperature to be converted: ");
    scanf("%f",&temp);
    printf("Enter an f if the temperature is in Fahrenheit");
    printf("\n or a c if the temperature is in Celsius: ");
    /*利用两个 printf 实现一句较长英文的换行输出*/
    scanf("\n%c",&tempType);
    if (tempType=='f')          /*输入字符'f'代表输入的是华氏温度*/
    {
        celsius=(5.0/9.0)*(temp-32.0);    /*根据数学公式进行温度转换*/
        printf("\nThe equivalent Celsius temperature is %.2fc.\n", celsius);
    }
    else
    {
        fahren=(9.0/5.0)*temp+32.0;      /*根据数学公式进行温度转换*/
        printf("\nThe equivalent Fahrenheit temperature is %.2fF.\n", fahren);
    }
    return 0;
}
```

程序运行时,输入温度 100,温度类型为 f,则说明是将华氏温度转换为摄氏温度,程序运行结果如图 5-21 所示。

```

输入一个温度: 100
输入一个代表温度的字符
(f:华氏温度,c:摄氏温度): f
摄氏温度为: 37.78C.
Press any key to continue

```

图 5-21 运行结果

本案例首先判断是哪一种情况, 如果输入字符 f, 则代表输入的是华氏温度, 把它转换成摄氏温度; 否则输入的是摄氏温度, 把它转换成华氏温度。

【案例 5-13】 输入一个年份, 判断该年是否为闰年。

案例分析: 判断闰年的条件为: ①年份数能被 4 整除, 但不能被 100 整除, 是闰年; ②年份数能被 400 整除, 是闰年。不满足这两个条件的不是闰年。以变量 leap 代表是否为闰年的信息。若为闰年, 令 leap=1; 若为非闰年, leap=0。

```

#include<stdio.h>
int main()
{
    int year, leap=0;
    printf("input year:");
    scanf("%d", &year);
    if(year%4==0)
    {
        if(year%100==0)
        {
            if(year%400==0)
                leap=1;
            else
                leap=0;
        }
        else
            leap=1;
    }
    else
        leap=0;
    if (leap)
        printf("%d is a leap year.\n", year);
    else
        printf("%d is not a leap year.\n", year);
    return 0;
}

```

输入年份 2000, 运行结果如图 5-22 所示。

本案例中, 由于判断闰年的条件有多个, 所以程序使用了 if 嵌套, 显得不是很直观。其实, if 嵌套经常可以用复合的关系或逻辑表示来实现, 如本案例的 if 嵌套就可以使用如下语句代替:

```

if ((year%4==0&&year%100!=0)|| (year%400==0))
    leap=1;
else
    leap=0;

```

```

input year:2000
2000 is a leap year.
Press any key to continue

```

图 5-22 运行结果

【案例 5-14】 编程实现运输公司计算用户运费。路程(s)越远, 每千米运费越低。标准如下:

$s < 250\text{km}$	没有折扣
$250\text{km} \leq s < 500\text{km}$	2%折扣
$500\text{km} \leq s < 1000\text{km}$	5%折扣
$1000\text{km} \leq s < 2000\text{km}$	8%折扣
$2000\text{km} \leq s < 3000\text{km}$	10%折扣
$3000\text{km} \leq s$	15%折扣

设每千米每吨货物的基本运费为 p ，货物重为 w ，距离为 s ，折扣为 d ，则总运费 f 的计算公式为： $f = p \times w \times s \times (1 - d)$ 。

分析此问题，可以看出，折扣的变化是有规律的，“变化点”都是 250 的倍数 (250、500、1000、2000、3000)。利用这一特点，可以在横轴上加一坐标 c ， c 的值为 $s/250$ 。 c 代表 250 的倍数。当 $c < 1$ 时，表示 $s < 250$ ，无折扣；当 $1 \leq c < 2$ 时，表示 $250 \leq s < 500$ ，折扣 $d = 2\%$ ；当 $2 \leq c < 4$ 时， $d = 5\%$ ；当 $4 \leq c < 8$ 时， $d = 8\%$ ；当 $8 \leq c < 12$ 时， $d = 10\%$ ；当 $c \geq 12$ 时， $d = 15\%$ 。

```
#include<stdio.h>
int main()
{
    int c,s;
    float p,w,d,f;
    printf("输入基本运费，货物重量，距离:");
    scanf("%f,%f,%d",&p,&w,&s);
    if (s>=3000)
        c=12;
    else
        c=s/250;
    switch(c)
    {
        case 0:d=0;break;
        case 1:d=2;break;
        case 2:
        case 3:d=5;break;
        case 4:
        case 5:
        case 6:
        case 7:d=8;break;
        case 8:
        case 9:
        case 10:
        case 11:d=10;break;
        case 12:d=15;break;
        default: printf("error"); break;
    }
    f=p*w*s*(1-d/100.0);
    printf("freight=%-15.4f\n",f);
    return 0;
}
```

程序运行结果如图 5-23 所示。

```
输入基本运费, 货物重量, 距离:100.5.5.80
freight=44000.0000
Press any key to continue
```

图 5-23 运行结果

5.4 循环结构

在进行算法设计时, 仅仅使用前面学过的顺序结构和选择结构, 往往解决不了较复杂的问题, 很多问题中的动作可能要多次重复, 如累加, 求一个班学生的平均分等。这时, 我们就需要使用循环控制结构来设计算法, 利用循环结构可以解决复杂的、重复性的操作。循环结构的作用是使某段程序重复执行, 具体循环的次数会根据某个条件来决定。C 语言是支持循环结构的, C 语言的循环结构主要包括三种基本形式: while 语句、do...while 语句、for 语句。

5.4.1 while 语句

while 语句属于“当型”循环。“当型”循环是指在循环条件成立时, 程序就一直执行循环体语句。while 语句的一般形式如下:

```
while(表达式)
    循环体语句;
```

while 语句的执行过程为: 首先计算 while 后圆括号内的表达式, 当表达式的值为真(非 0)时, 执行循环体语句, 然后继续判断表达式的值, 重复上述执行过程, 只有当表达式为假(0)时才退出循环, 程序跳转到循环体后面的第 1 行代码处执行。while 语句流程图如图 5-24 所示。

说明:

- ① while 是关键字。while 后圆括号内的表达式一般是条件表达式或逻辑表达式, 但也可以是 C 语言中任意合法的表达式, 其计算结果为 0, 则跳出循环体; 非 0, 则执行循环体。
- ② 循环体语句可以是一条语句, 也可以是多条语句, 如果循环体语句包含多条语句, 则需要用一对花括号“{}”把循环体语句括起来, 采用复合语句的形式。

【案例 5-15】 求前 100 个自然数的和, 即: $\sum_{i=1}^{100} i$ 。

案例分析: 这是一个简单的求和问题, 需要连续累加, 因此只能使用循环结构实现重复累加的操作。设变量 sum 用于存放循环执行过程中的求和结果, 变量 i 为循环控制变量, 同时也是每一次求和运算的基本数据项, 然后可以利用 while 循环结构进行循环累加求和。

程序流程图如图 5-25 所示。根据流程图编写程序代码如下:

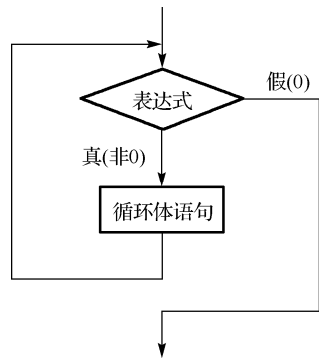


图 5-24 while 语句流程图

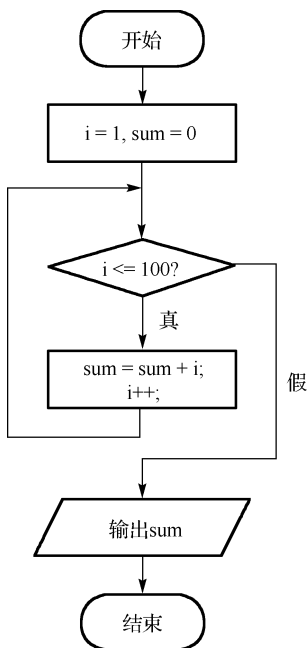


图 5-25 程序流程图

```

#include<stdio.h>
int main()
{
    int i=1,sum=0;           /*定义变量并初始化*/
    while (i<=100)
    {
        sum=sum+i;          /*累加求和*/
        i++;                /*修改基本数据项*/
    }
    printf("sum=%d\n",sum);
    return 0;
}

```

程序运行结果为：sum=5050。

在读程序时，正确分析语句的执行顺序，即正确判断语句的跳转及确定此时变量的值是非常重要的，是能否正确理解程序的关键，表 5-1 是对案例 5-15 程序执行过程的具体分析。

表 5-1 程序执行过程的具体分析

执行顺序	执行语句	执行结果	sum 的值	i 的值	说 明
1	i=1;sum=0;		0	1	变量赋初值
2	计算表达式 i<=100	1<=100 结果为“真”			判断循环条件
3	sum=sum+i; i++;	sum←0+1 i←1+1	1	2	执行循环体语句
4	计算表达式 i<=100	2<=100 结果为“真”			判断循环条件
5	sum=sum+i; i++;	sum←1+2 i←2+1	3	3	执行循环体语句

续表

执行顺序	执行语句	执行结果	sum 的值	i 的值	说 明
...	4950	100	...
200	计算表达式 $i \leq 100$	$100 \leq 100$ 结果为“真”			判断循环条件
201	$sum = sum + i;$ $i++;$	$sum \leftarrow 4950 + 100$ $i \leftarrow 100 + 1$	5050	101	执行循环体语句
202	计算表达式 $i \leq 100$	$101 \leq 100$ 结果为“假”			判断循环条件
203	$printf("sum = \%d \n", sum);$				退出循环体, 执行循环体下面的语句

需要注意以下几个问题。

① 累加求和算法。此程序采用的算法思想称为累加求和，即：不断用新累加的值取代变量的旧值，最终得到求和结果，变量 `sum` 也叫“累加器”，初值一般为 0。累加求和尽管方法简单，但却是循环结构程序设计中经常采用的一种算法思想，后面的很多复杂程序最终都可以转换为累加求和或类似累加求和的问题来解决。使用 C 语言的循环结构对若干数进行累加求和一般要包括以下几个步骤：

- 步骤 1：设置基本数据项的初值(如上面程序中的 $i=1$)；
- 步骤 2：设置存放结果变量的初值(如上面程序中的 $sum=0$)；
- 步骤 3：循环条件判断，若条件满足转步骤 4，否则转步骤 6；
- 步骤 4：累加并修改基本数据项(如上面程序中的 $sum=sum+i;i++$)；
- 步骤 5：转步骤 3；
- 步骤 6：结束并输出结果。

② 必须给变量赋初值。在 C 语言中定义的变量必须要赋初值，即使变量的初值为 0，赋初值也不能省略。如果没有给变量赋初值，那么变量的初值就会是一个不可预知的数，结果将没有意义。例如本题中，读者可以省略赋值语句“`sum=0;`”来测试运行结果是否正确。

③ 正确判断条件的边界值。当 i 的值为 100 时，程序将继续执行循环体，然后控制流程再次判断条件表达式，此时， i 的值为 101(见表 5-1)，表达式“ $i \leq 100$ ”的结果为假，退出循环。退出循环后，循环控制变量 i 的值是 101，而不是 100。

④ 避免出现“死循环”。使用 `while` 循环一定要注意在循环体语句中出现修改循环控制变量的语句，使循环趋于结束，如本例中的“ $i++$ ；”，否则条件表达式的计算结果永远为“真”，就会出现死循环。

⑤ 可能出现循环体不执行的情况。`while` 循环是先判断表达式的值，后执行循环体，因此，如果一开始表达式为假，则循环体一次也不执行。

⑥ `while` 后面圆括号内的表达式一般为关系表达式或逻辑表达式，但也可以是其他类型的表达式，如算术表达式等。只要表达式运算结果为非 0，就表示条件判断为“真”，运算结果为 0，就表示条件判断为“假”。如下面的几种循环结构，它们反映的逻辑执行过程是等价的，均表示当 n 为奇数时执行循环体，否则退出循环。

<pre>while(n%2) { ... }</pre>	<pre>while(n%2==1) { ... }</pre>	<pre>while(n%2!=0) { ... }</pre>
-----------------------------------	--------------------------------------	--------------------------------------

有时，条件表达式可能只是一个变量，如以下程序段：

```
...
P=1;
while(P)
{
    ...
    P=0;
    ...
}
```

【案例 5-16】 使用 while 语句求 $n!$ ， $n!=1\times 2\times 3\times \cdots \times n$ 。

案例分析：该案例与案例 5-15 非常相似，只是把求和改成求乘积。另外，由于 n 的值并不确定，需要程序执行的时候由用户输入，所以要用到输入函数。本案例使用变量 `sum` 存储乘积值，由于阶乘的数量级递增非常快，采用整型数据类型存储数量级很有限，为了能容纳较大的阶乘值，将 `sum` 定义为 `double` 类型，尽管如此，在测试程序时还是要注意数据的溢出问题。

```
#include<stdio.h>
int main()
{
    int n,i=1;
    double sum=1;
    printf("请输入一个正整数: ");
    scanf("%d",&n);
    while (i<=n)
    {
        sum=sum*i;    /*累乘求积*/
        i++;          /*修改基本数据项 i*/
    }
    printf("%d!=%.0f\n",n,sum);
    return 0;
}
```

运行程序时，输入值为 10，程序运行结果如图 5-26 所示。



```
请输入一个正整数: 10
10!=3628800
Press any key to continue
```

图 5-26 运行结果

循环变量赋初值、判断控制表达式和修改循环变量称“循环三要素”。一般来说，进入循环之前，应该给循环变量赋初值，确保循环能够正常开始；在控制表达式中判断循环变量是否达到循环终止值；在循环体中对循环变量进行修改，以使循环正常趋向终止。在编写程序时要注意它们的位置关系。循环控制变量的初值可能会影响控制表达式的设计和控制变量修改语句的语序。

此案例虽然与案例 5-15 非常相近，但仍有两个需要注意的问题。

① 变量合理赋初值。变量初值的选取要根据实际情况,本案例中用来存放乘积结果的变量 `sum` 的初值就应赋 1,而不是 0。

② 防止出现数据溢出错误。累乘结果变量 `sum` 的结果虽然是整数,在这里不能定义成 `int` 型数据。由于 `int` 型变量可以存放数据的范围比较有限(根据编译环境的不同而不同),当用户输入的值比较大时,就可能得到一个非常大的结果,为防止在计算阶乘时发生数据溢出错误,应把 `sum` 定义成 `double` 类型(但还是要注意输入的数据不能太大)。

【案例 5-17】 编写程序,输入一个字符序列,直至换行为止,统计出字母、数字、空格和其他字符的个数。

案例分析:这是一个关于字符处理的问题,首先可以定义一个字符变量 `ch`,利用 `getchar()` 函数把用户从键盘输入的字符逐个接收,存储在 `ch` 中,然后对 `ch` 进行判断分类。当读取的字符不是换行符时重复执行循环体,直到遇到换行符为止。`while` 语句的条件表达式可以写成“`ch!='\n'`”。

```
#include<stdio.h>
int main()
{
    char ch;
    int letter,digital,space,other;
    letter=digital=space=other=0;          /*各变量赋初值 0*/
    while((ch=getchar())!='\n')
        /*接收从键盘输入的字符,并判断是否为换行符,遇到换行符则停止循环*/
    {
        if(ch>='A'&&ch<='Z' || ch>='a' && ch<='z') letter++;
        /*判断是否为大写字母*/
        else if(ch>='0'&&ch<='9') digital++; /*判断是否为数字*/
        else if(ch==' ') space++;           /*判断是否为空格*/
        else other++;
    }
    printf("letter=%d,digital=%d,space=%d,other=%d\n",letter,digital,
        space,other);
    return 0;
}
```

程序运行结果如图 5-27 所示。

```
OID<>8U0IU<8 >G<D* >098s >9
letter=10,digital=6,space=4,other=8
Press any key to continue
```

图 5-27 运行结果

注意:

表达式 `((ch=getchar())!='\n')` 的执行分两步,首先利用 `getchar()` 函数从终端接收一个字符,存储在 `ch` 中,然后再判断 `ch` 是否为 `'\n'`,不能省略内部的括号。如果写成如下形式:

```
(ch=getchar())!='\n')
```

调试结果就会发生错误，因为表达式的关系运算符“!=”的运算优先级别高于赋值运算符“=”，程序中语句相当于：

```
while(ch=(getchar()!='\n'))
```

即先把接收的字符与'\n'进行关系运算，再把关系运算的结果“真”或者“假”存储在 ch 中，这显然是错误的。

5.4.2 do...while 语句

do...while 语句属于“直到型”循环，循环体语句一直循环执行，直到循环条件表达式的值为假时停止，语句的一般形式如下：

```
do
    循环体语句
while(表达式);
```

执行过程：先执行循环体语句，然后计算 while 后圆括号内的表达式，当表达式为“真”（非 0）时，则再次执行循环体语句，重复上述操作直到表达式为“假”（0）时退出循环。其中循环体语句可以是简单语句，也可以是用一对花括号“{}”括起来的复合语句。do...while 语句流程图如图 5-28 所示。

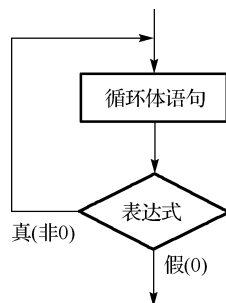


图 5-28 do...while 语句流程图

说明：

① do...while 语句中“while(表达式);”的分号是不能省略的，这一点与 while 语句不一样。

② do...while 语句是先执行循环体语句，后判断表达式，因此无论条件是否成立，将至少执行一次循环体。而 while 语句是先判断表达式，后执行循环体语句，因此，如果表达式在第一次判断时就不成立，则循环体一次也不执行。

while 语句和 do...while 语句的比较如下：

一般来说，对于同一个问题，使用 while 语句或 do...while 语句的结果是一样的，也就是说，只要循环体相同，其结果也会相同。比如求 1~100 的和，分别使用 do...while 和 while 语句的形式如下。

<pre>int i=1,sum=0; do { sum=sum+i; i++; }while (i<=100); printf("sum=%d\n",sum);</pre>	<pre>int i=1,sum=0; while(i<=100) { sum=sum+i; i++; } printf("sum=%d\n",sum);</pre>
--	--

从上面两个程序段看，除了循环判断条件所处的位置不同，其他并没有什么区别。事实上，二者并不完全等价，如果第一次进行循环时，while 后面的表达式就不成立，那么对于 while 循环来说，循环体语句一次也不执行，程序直接跳过循环结构，执行下面的语句；对于 do...while 循环来说，循环体语句需要执行一次才会跳出循环结构。

【案例 5-18】 编写程序，实现对用户输入口令的校验。假设用户预存口令是 A，若用户输入的口令和预设口令不一致，则需要重新输入，直到与预设口令一致时为止。

案例分析：定义一个字符型变量 c 用来存放用户输入的口令。循环的条件是，用户输入的口令和预设的口令不一致。用户需要先输入口令再进行判断，因此选用 do...while 循环更合适。

```
#include<stdio.h>
int main()
{
    char c;
    printf("请输入口令: \n");
    do
    {
        c=getchar();          /*接收用户输入的口令*/
    }while(c!='A');          /*假定预设口令是字符'A'*/
    printf("校验成功\n");
    return 0;
}
```

92



图 5-29 运行结果

程序运行结果如图 5-29 所示。

【案例 5-19】 用公式 $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$ ，求 π 的近似值，直到最后一项的绝对值小于 10^{-6} 为止。

案例分析：本案例属于累加求和问题，可以定义浮点型变量 d 存放基本数据项，注意题目中相邻基本数据项的符号不同，因此定义变量 sign 表示当前数据项的符号，初值为正号，即 sign=1，每循环一次，都使 sign 的符号取反，即 sign=-sign，其他步骤与一般的累加求和问题相同。

```
#include<stdio.h>
#include<math.h>
int main()
{
    double n=1.0,d=1.0,pi=0.0;
    int sign=1;
    do
    {
        pi=pi+d;
        n=n+2;
        sign=-sign;          /*改变数据项的符号*/
        d=sign/n;           /*求出数据项*/
    }
    while(fabs(d)>=1.0e-6);
    pi=4.0*pi;
    printf("pi=%10.7f\n",pi);
    return 0;
}
```

程序输出的结果为： $\pi=3.1415907$ 。

注意：语句的先后顺序有时也非常重要，例如，案例 5-19 如果改写成如下形式：

```
#include<stdio.h>
#include<math.h>
int main()
{
    double n=1.0,d=1.0,pi=0.0;
    int sign=1;
    do
    {
        n=n+2;
        sign=-sign;
        d=sign/n;
        pi=pi+d;
    }
    while(fabs(d)>=1.0e-6);
    pi=4.0*pi;
    printf("pi=%10.7f\n",pi);
    return 0;
}
```

则程序运行的结果为： $\pi=-8584053$ 。结果显然不正确，只是修改了循环体中的一个语句的顺序，结果就会产生错误，如果将程序在此基础上进行进一步的修改，如下：

```
double n=1.0,d=1.0,pi=1.0;
int sign=1;
do
{
    n=n+2;
    sign=-sign;
    d=sign/n;
    pi=pi+d;
}
while(fabs(d)>=1.0e-6);
pi=pi-d;
pi=4.0*pi;
printf("pi=%10.7f\n",pi);
return 0;
```

则程序运行的结果与先前程序运行的结果一致，即输出结果为： $\pi=3.1415907$ 。由此可以看出，变量初值改变了，循环体中语句的顺序就要做出相应的调整，同时循环的次数可能也会受到影响，在编写程序时一定要考虑这些因素。请思考：为什么在循环体后要添加语句“ $\pi=\pi-d$;”？

5.4.3 for 语句

for 语句是循环控制结构中使用最为广泛的一种控制语句，它充分体现了 C 语言的灵活性。

for 语句有时也被称为“计数型”循环，因为它特别适合于已知循环次数的情况。但事实上，for 循环同样适用于循环次数不确定而只知道循环结束条件的情况。for 循环可以实现所有的循环问题，它是 C 语言中形式最灵活，功能最强大的一种循环控制结构。

for 语句的一般形式如下：

```
for(表达式 1; 表达式 2; 表达式 3)
    循环体语句;
```

从语法形式上看，for 语句的语法要比 while 语句复杂，for 后面的圆括号内有三个表达式，并使用分号“;”分隔，这三个表达式的运算次数、运算时间及在循环中发挥的作用各不相同。它的执行过程如下。

步骤 1：计算表达式 1。

步骤 2：计算表达式 2，若表达式 2 的值为“真”（非 0），则执行一次循环体语句，然后转步骤 3，若表达式 2 的值为“假”（0），则转步骤 4。

步骤 3：计算表达式 3，然后转步骤 2。

步骤 4：退出循环，执行 for 语句后面的其他语句。

for 语句流程图如图 5-30 所示。

其中表达式 1、表达式 2 和表达式 3 可以是任何一种 C 语言合法的表达式，但最常用、最简单的形式是：在表达式 1 中给循环变量赋初值；表达式 2 是循环条件控制表达式；表达式 3 实现循环控制变量的改变，使循环趋于结束。具体如下：

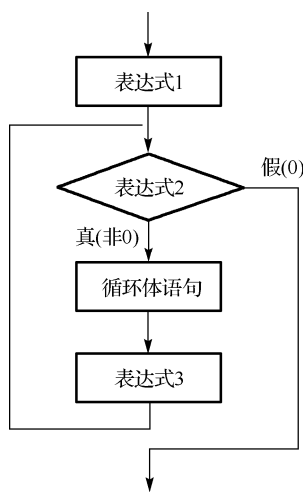


图 5-30 for 语句流程图

```
for(循环变量赋初值; 循环条件; 循环变量增值)
```

for 语句的功能等价于下面 while 语句：

```
表达式 1;
while(表达式 2)
{
    循环体语句;
    表达式 3;
}
```

如果用 for 语句改写案例 5-15，可以改写为以下语句：

```
#include<stdio.h>
int main()
{
    int i,sum=0;
    for(i=1;i<=100;i++)
        sum=sum+i;
    printf("sum=%d\n",sum);
    return 0;
}
```


由此可以看出，相对于 while 语句，for 语句在形式上更加简洁、方便。

【案例 5-20】 设 $n=30$ ，编写程序，计算并输出 $S(n)$ 的值。 $S(n)=(1\times 2)/(3\times 4)-(3\times 4)/(5\times 6)+(5\times 6)/(7\times 8)+\cdots+(-1)^{(n-1)}\times[(2n-1)\times 2n]/[(2n+1)\times (2n+2)]+\cdots$ 。

案例分析：这是一个累加求和的问题，题目明确是求 30 项的和，因此选用 for 循环是最合适的。在程序中设变量 s 用于存放循环执行过程中的求和结果，设变量 i 为循环控制变量，每一次求和运算的数据项由运算表达式给出，数据项的值会随循环变量 i 的改变而改变。设计出程序流程图如图 5-31 所示，根据流程图编写程序代码如下：

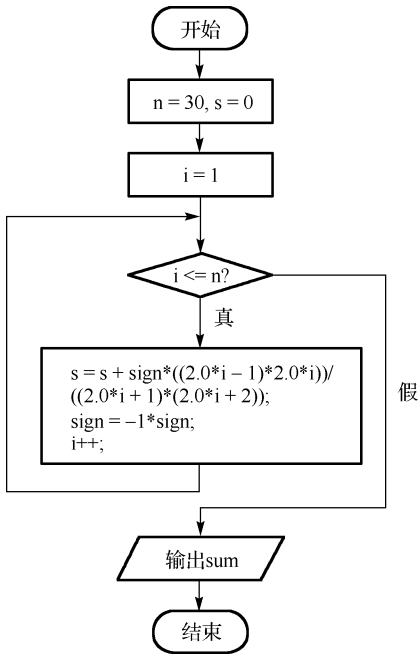


图 5-31 程序流程图

```

#include<stdio.h>
int main()
{
    int n=30,i,sign=1;
    double s=0;
    for(i=1;i<=30;i++)
    {
        s=s+sign*((2.0*i-1)*(2.0*i))/((2.0*i+1)*(2.0*i+2));
        sign=-1*sign;        /*改变数据项的符号*/
    }
    printf("S(30)=%.6lf\n",s);
    return 0;
}
    
```

程序运行结果为： $S(30)=-0.459873$ 。

关于 for 语句的几点说明如下。

① 循环体语句可以是简单语句，也可以是使用一对花括号括起来的复合语句。如果是一

个语句，也可以和 for 写在一行上，这样使程序看起来更加简洁；如果循环体包含多条语句，最好另起一行，采用一对花括号括起来的复合语句形式，增加程序的可读性。

② 表达式的省略。for 语句中的 3 个表达式均可以省略，但是 2 个分号不能省略。

● 省略表达式 1

如果 for 语句中的表达式 1 被省略，则表达式 1 的内容可以放在 for 循环结构之前。表达式 1 的内容一般来说是给循环变量赋初值，那么如果在循环结构之前的程序中循环变量已经有初值，那么表达式 1 就可以省略，但分号不能省。如案例 5-20 中，for 语句中如果省略表达式 1，可以改写成如下形式：

```
...
int n=30,i=1,sign=1;
double s=0;
for(;i<=30;i++)
{
    s=s+sign*((2.0*i-1)*(2.0*i))/((2.0*i+1)*(2.0*i+2));
    sign=-1*sign;    /*改变数据项的符号*/
}
printf("S(30)=%.6lf\n",s);
...
```

96

● 省略表达式 2

如果表达式 2 省略，就意味着每次执行循环体之前不用判断循环条件，循环就会无休止地执行下去，就有可能形成了“死循环”，编程中要避免此类情况出现。

● 省略表达式 3

如果表达式 3 省略，则必须在程序中另外添加修改循环变量值的语句，保证循环能够正常结束。案例 5-20 如果省略表达式 3，程序可以改写成如下形式：

```
int n=30,i,sign=1;
double s=0;
for(i=1;i<=30;)
{
    s=s+sign*((2.0*i-1)*(2.0*i))/((2.0*i+1)*(2.0*i+2));
    sign=-1*sign;    /*改变数据项的符号*/
    i++;
}
printf("S(30)=%.6lf\n",s);
```

● 同时省略表达式 1 和表达式 3

如果表达式 1 和表达式 3 同时省略，只有表达式 2，也就是说只有循环条件，那就和 while 循环功能一样。下面两段程序是等价的。

```
while (i<=100)
{
    sum=sum+i;
    i++;
}
```

等价于：

```
for (;i<=100;)
{
    sum=sum+i;
    i++;
}
```

● 同时省略3个表达式

当然，for 循环的3个表达式也可同时省略，即：

```
for (;;)
{
    ...
}
```

这种形式就会使循环体一直执行下去，形成“死循环”。

③ 表达式1和表达式3可以和循环变量无关。前面讲到，一般来说表达式1是给循环变量赋初值，表达式3是修改循环变量的值。但表达式1和表达式3的内容也可以和循环变量完全无关。

例如，用for语句实现求整数1~100的和，程序如下：

```
int i,sum=0;
for (i=1;i<=100;i++) sum=sum+i;
printf("sum=%d\n",sum);
```

也可以写成：

```
int sum,i=0;
for(sum=0;i<100;sum=sum+i) i++;
printf("sum=%d\n",sum);
```

第2种形式虽然结果也正确，但和第1种形式相比，程序的可读性和可维护性就大大降低了。可见，虽然for语句使用起来形式非常灵活，但是一般来说还是要遵从常用的形式，不要在表达式1和表达式3中出现和循环控制变量无关的内容。

④ 表达式1和表达式3可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号隔开。在以后的学习中我们会遇到一些较为复杂的问题，与循环控制相关的变量可能有一个以上，如以下程序段：

```
for(i=0,j=10;i<=j;i++,j--)
{
    ...
}
```

表示在循环之初，分别对i和j赋初值0和10，每一趟循环结束时，分别对i增1，对j减1。

【案例5-21】 编写程序输出所有的水仙花数。水仙花数是指一个3位数，其各位数字的立方和等于该数本身。例如： $153=1^3+5^3+3^3$ ，所以153就是水仙花数。

案例分析：这是一个典型的穷举算法，因为水仙花数是一个3位数，程序需要对所有的3

位数都做判断，所以在程序中可以定义一个变量 i ，使 i 在 100~999 之间循环，逐个判断 i 是否为水仙花数。在这类问题中，循环变量 i 有明确的初值和终值，并且是递增或递减变化的，选用 for 循环最合适。另外，题目中要求求各位数字的立方和，这种问题常通过 “/” 和 “%” 两种运算结合使用的方式来解决。

```
#include<stdio.h>
int main()
{
    int a,b,c,i;
    for(i=100;i<=999;i++)
    {
        a=i/100;          /*求出 i 的百位数字*/
        b=i/10%10;        /*求出 i 的十位数字*/
        c=i%10;           /*求出 i 的个位数字*/
        if(i==a*a*a+b*b*b+c*c*c)
            printf("%d 是水仙花数.\n",i);
    }
    return 0;
}
```

程序运行结果如图 5-32 所示。

注意：

① 在计算机解决实际问题时，常常会用到类似本程序的“穷举法”。“穷举法”解决的问题一般具有如下特点：如果问题

有解，为一组或多组，必定全在某个集合中；如果这个集合内无解，集合外也肯定无解。这样，在解决问题时，就可以将集合中的元素一一列举出来，验证是否为问题的解。本案例就是一一验证 100~999 之间的所有数，最终找出答案。

② 程序中在进行相等关系判断 ($i==a*a*a+b*b*b+c*c*c$) 时，使用了关系运算符 “==”，而不是 “=”，后者是赋值运算符，在 C 语言中这两种运算符形式是不一样的，要注意区别。

前面介绍的三种循环语句 while、do...while 和 for 形式虽然不同，但主要结构成分都是循环三要素。三种语句都可以实现循环，一般来说，可以互相替代。但它们也有一定的区别，使用时应根据语句的特点和实际问题的需要选择合适的语句，它们的区别和特点如下。

① while 和 do...while 语句一般实现条件循环，即无法预知循环的次数，循环只是在一定条件下进行的。而 for 语句大多实现计数式循环。

② 一般来说，while 和 do...while 语句的循环变量赋初值在循环语句之前，循环结束条件是 while 后面圆括号内的表达式，循环体中包含循环变量修改语句。一般 for 循环则是循环三要素在一行。因此，for 循环语句形式更简洁，使用更灵活。

③ while 和 for 语句的循环是先测试循环条件，后执行循环体语句，循环体可能一次也不执行。而 do...while 语句是先执行循环体语句，后测试循环条件，所以循环体至少被执行一次。

知道了三种循环各自的特点，在实际使用时就要根据这些特点合理选择。



```
153是水仙花数.
370是水仙花数.
371是水仙花数.
407是水仙花数.
Press any key to continue
```

图 5-32 运行结果

5.4.4 break 语句和 continue 语句

前面学习了 C 语言的三种循环结构：while 语句、do...while 语句和 for 语句，这三种形式都是在一定的条件满足时进行循环的，当循环条件不满足时，循环才会结束。但有的实例中允许在意外情况下，中途结束循环。C 语言提供了 break 语句和 continue 语句改变控制流。在循环体中的 break 语句可实现终结本轮循环的执行，continue 语句可控制结束本次循环的执行。

1. break 语句

在前面的学习中 break 语句可以使流程跳出 switch 结构，同时它也可以用在 while 语句、do...while 语句和 for 语句中，用于跳出循环结构。当 break 用于这三种循环语句时，可使程序跳出本层循环结构，接着执行循环体下面的语句。其一般形式如下：

```
break;
```

【案例 5-22】 分别输出半径为 1~10 中整数的圆的面积，要求当圆的面积大于 100 时停止输出。

案例分析：圆的半径从 1 增加到 10，每次增加 1。圆的面积计算方法每次都相同，所以可以使用循环实现。半径从 1 到 10 共 10 次，可使用 for 循环。当计算的圆面积大于 100 时，提前结束循环，需要使用 break 语句。设计程序流程图如图 5-33 所示，根据流程图编写程序代码如下：

```
#include<stdio.h>
int main()
{
    int r;
    double area,pi=3.14159;
    for(r=1;r<=10;r++)
    {
        area=pi*r*r;
        if(area>100)
            break;
        printf("r=%d,area=%.6lf\n",r,area);
    }
    return 0;
}
```

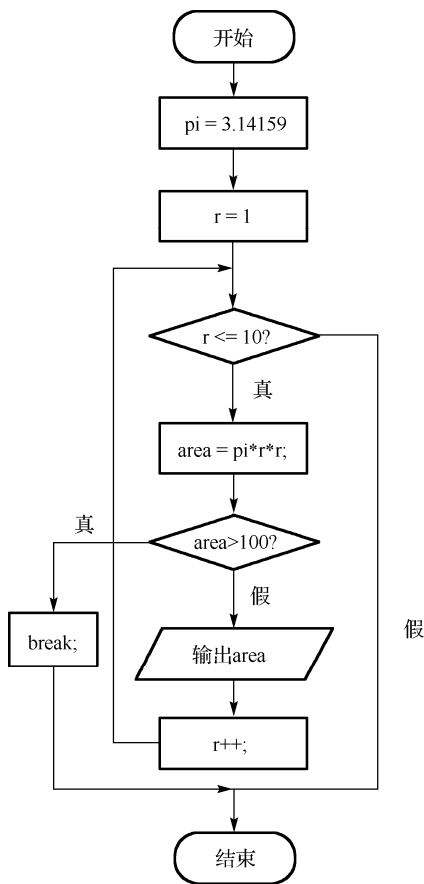


图 5-33 程序流程图

程序运行结果如图 5-34 所示。

```
r=1,area=3.141590
r=2,area=12.566360
r=3,area=28.274310
r=4,area=50.265440
r=5,area=78.539750
Press any key to continue
```

图 5-34 运行结果

当 $r=6$ 时, 条件 $\text{area}>100$ 为真, 执行 `break` 语句, 提前结束循环, 即不再继续执行其余的几次循环。程序跳转到 `for` 循环下面的语句继续执行。

说明:

① `break` 语句只能用于 `while`、`do...while` 和 `for` 循环语句, 以及 `switch` 语句中, 不能用于其他语句。

② 如果 `break` 语句用在多重循环结构体中, 使用 `break` 语句只能使程序退出 `break` 语句所在的最内层循环。

2. continue 语句

`continue` 语句的作用是结束本次循环, 即跳过循环体中下面尚未执行的语句, 接着进行下一次是否执行循环体的判断。其一般形式如下:

```
continue;
```

`continue` 语句只能用在循环结构中。

对于 `while` 和 `do...while` 语句, `continue` 语句使程序结束本次循环, 跳转到循环条件的判断部分, 根据条件判断是否进行下一次循环; 对于 `for` 语句, `continue` 语句使程序不再执行循环体中下面尚未执行的语句, 直接跳转去执行“表达式 3”, 然后再对循环条件“表达式 2”进行判断, 根据条件判断是否进行下一次循环。

【案例 5-23】 编写程序, 实现输入若干学生的成绩, 求平均成绩。

```
#include<stdio.h>
int main()
{
    int i,n,score;
    float sum=0,aver;
    printf("请输入学生的个数:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("请输入学生的成绩:");
        scanf("%d",&score);
        if(score<0||score>100)           /*学生成绩输入有误*/
        {
            printf("输入成绩有误, 请重新输入!\n");
            i--;                          /*此次输入成绩不算, 计数应减去 1*/
            continue;
        }
        sum=sum+score;
    }
    aver=sum/n;
    printf("average=%.2f\n",aver);
    return 0;
}
```

程序运行结果如图 5-35 所示。

当程序执行时, 用户输入的成绩如果不在 0~100 之间, 即 if 语句的条件 “score<0||score>100” 成立, 程序就会输出错误信息, 计数变量 i 减 1, 执行 continue 语句, 这时, 程序就会结束本次循环, 不再执行循环体中下面尚未执行的语句 “sum=sum+score;”, 直接跳转去执行 “表达式 3” (i++), 接着判断 “表达式 2” (i<=n), 决定是否进行下一次循环。

continue 语句和 break 语句的区别是: continue 语句只是结束本次循环, 而不是终止整个循环的执行; 而 break 语句则是结束整个当前所在循环过程, 执行循环体后面的语句。

```

请输入学生的个数:5
请输入学生的成绩:87
请输入学生的成绩:101
输入成绩有误, 请重新输入!
请输入学生的成绩:1001
输入成绩有误, 请重新输入!
请输入学生的成绩:100
请输入学生的成绩:80
请输入学生的成绩:76
请输入学生的成绩:92
average=87.00
Press any key to continue
  
```

图 5-35 运行结果

5.4.5 多重循环结构

在处理实际问题时, 有时仅仅使用前面学过的循环是不够的, 在已有循环结构的循环体语句中还需要包含循环结构, 这就是多重循环。

一个程序中的多个循环语句之间存在两种关系: 并列关系和嵌套关系。循环不允许有交叉, 循环之间的关系如图 5-36 所示。

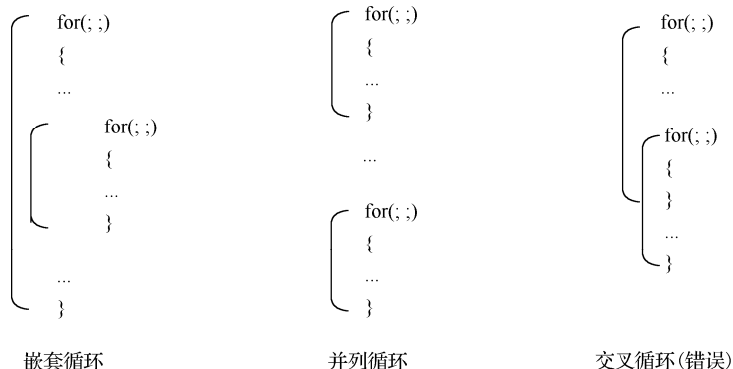


图 5-36 循环之间的关系

嵌套循环是指一个循环语句的循环体内完整地包含了另一个完整的循环结构。前述三种循环结构 (while 循环、do...while 循环、for 循环) 可以任意组合嵌套。它的执行过程是: 首先进行外层循环的条件判断, 当外层循环条件成立时, 顺序执行外层循环体语句, 遇到内层循环, 则进行内层循环条件判断, 并在内层循环条件成立的情况下反复执行内层循环体语句, 当内层循环因循环条件不成立而退出后, 重新返回到外层循环, 并顺序执行外层循环体的其他语句, 外层循环体执行一次后, 重新进行下一次外层循环条件判断, 若条件依然成立, 则重复上述过程, 直到外层循环条件不成立时, 退出双重循环嵌套, 执行后面其他语句。例如, 双重循环嵌套流程图如图 5-37 所示。

多重循环不仅包含双重循环结构, C 语言还允许循环结构的多重嵌套。如果一个循环的外面有两层循环就叫三重循环, 如图 5-38 所示就是一个三重循环结构。当然, 还允许有四重、五重等更多重的循环。理论上嵌套可以是无限的, 但一般使用两重或三重的比较多, 若嵌套层数太多, 就降低了程序的可读性和执行效率。

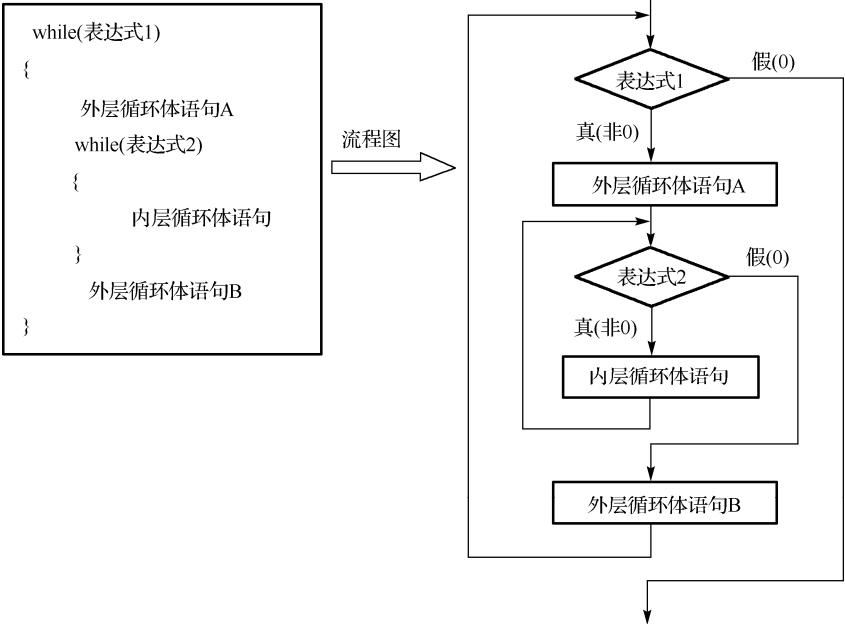


图 5-37 双重循环嵌套流程图

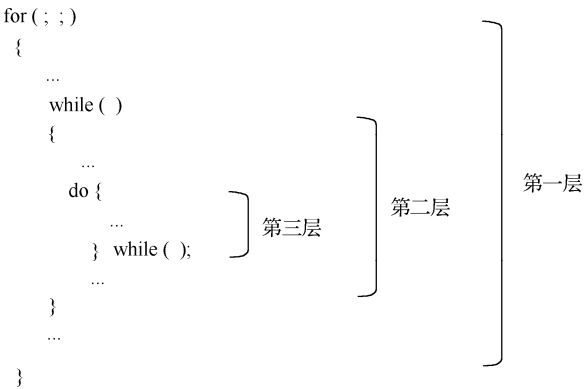


图 5-38 三重循环结构

【案例 5-24】 编写程序，输出 1000 以内所有的完数。如果一个整数的因子之和等于这个数本身，这个数就被称为完数。例如，6 的因子是 1、2、3，并且 1+2+3=6，所以 6 是完数。

案例分析：此案例可分成两步实现。

第 1 步：在程序中判断一个数 n 是否为完数。在编写程序时，可以定义一个变量 s 作为“累加器”，用 n 逐一去除 $1\sim n-1$ ，如果能除尽，就说明是 n 的因子，把它累加到 s 上，这是我们前面讲过的“穷举法”，可以选用 for 循环实现。

第 2 步：外层循环对 1000 以内的所有正整数一一进行判断，利用第 1 步的方法，逐个判断 n 的因子之和 s 是否等于 n 。若相等，则显示输出。对 1000 以内的所有正整数进行一一测试，同样使用“穷举法”，选用 for 循环结构。

程序设计思路如下：

```
for(n=2;n<=1000;n++)
```



```

{
    求 n 的所有因子之和赋给 s;
    若 n==s, 则 n 是完数, 显示输出;
}

```

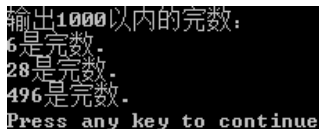
将上面的内循环改成代码, 程序如下:

```

#include<stdio.h>
int main()
{
    int i,n,s;
    printf("输出 1000 以内的完数: \n");
    for(n=2;n<=1000;n++)          /*外循环, 对 2~1000 之间的数进行判断*/
    {
        s=0;
        for(i=1;i<n;i++)          /*内循环, 求出 n 的所有因子之和*/
            if(n%i==0)
                s+=i;
        if(n==s)                  /*判断 a 是否等于所有因子之和*/
            printf("%d 是完数.\n",n);
    }
    return 0;
}

```

程序运行结果如图 5-39 所示。



```

输出1000以内的完数:
6是完数.
28是完数.
496是完数.
Press any key to continue

```

图 5-39 运行结果

【案例 5-25】 编写程序, 打印九九乘法口诀表。

案例分析: 乘法口诀表的形式如下。

```

1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
...
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81

```

九九乘法口诀表是一个二维图表, 这种表的处理常采用双重循环来实现。外循环控制输出行, 内循环控制输出某行中的具体内容(即列)。求解此类问题的关键是分析图表的规律, 九九乘法表的规律如下。

① 乘法表共有 9 行。用外循环控制行, 是定数循环, 选用 for 循环比较合适。

② 每行算式个数规律: 第几行就有几列算式。用内层循环输出每行的算式, 内循环每执行一次, 输出一个算式, 因此内循环执行次数=外循环变量的值。内循环每次也是定数循环, 选用 for 循环。

③ 每个算式都既与所在行有关，又与所在列有关，规律是：列*行=积。
根据分析，编写程序代码如下：

```
#include<stdio.h>
int main()
{
    int i,j;
    for(i=1;i<=9;i++)                /*外循环控制输出行*/
    {
        for(j=1;j<=i;j++)            /*输出该行的内容*/
            printf("%d*%d=%-3d",j,i,i*j);
        printf("\n");                /*每行结束后，输出换行*/
    }
    return 0;
}
```

程序运行结果如图 5-40 所示。

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
Press any key to continue
```

图 5-40 运行结果

注意：如果是多重循环，外循环和内循环应选用不同的循环控制变量。

5.4.6 循环结构程序举例

【案例 5-26】 编写程序，判断程序中输入的 m 是否为素数。

案例分析：所谓素数，也叫质数，就是一个正整数，除本身和 1 以外并没有任何其他因子。例如 2、3、5、7 就是素数。

方法一：

算法分析：若 m 有因子，则因子一定是成对出现的，因此判断的范围可以限定到 2 及以上，若此区间内存在能整除 m 的整数，则 m 就不是素数，若此区间内没有一个能整除 m 的数，说明 m 除本身和 1 以外并没有任何其他因子，则 m 是素数。

算法设计：在程序中定义一个整数 i 作为循环变量，定义一个整数 k ，让 i 从 2 到 k 循环，判断如果 m 能被 i 整除，则提前结束循环，此时 i 必然小于或等于 k ；如果 m 一直没有被 2 到 k 之间的任何一个整数整除，在最后完成一次循环后， i 还要加 1，即 $i=k+1(>k)$ ，循环才能正常终止。因此，在循环结束之后判断 i 的值是否大于 k ，若是，则表明 m 未曾被 2 到 k 之间的任意一个整数整除过，因此 m 就是素数，否则， m 就不是素数。程序如下：

```
#include<stdio.h>
#include<math.h>
```

```

int main()
{
    int m,i,k;
    scanf("%d",&m);
    k=sqrt(m);
    for(i=2;i<=k;i++)
        if(m%i==0) break;          /*m 能被某个 i 整除, 已不是素数, 结束循环*/
    if(i>k)
        printf("%d is a prime number\n",m);
        /*结束循环后, 对 i 的值做判断, 若循环正常结束, 则 i 的值一定是大于终值
        k 的, 这时说明循环中从来没有出现过(m%i==0)为真的情况, 即 m 是素数;
        若 i 的值不大于 k, 则说明循环是中途从 break 语句退出的, 即 m 不是素数*/
    else
        printf("%d is not a prime number\n",m);
    return 0;
}

```

运行程序, 如果输入数据 25, 则输出为 “25 is not a prime number”, 如果输入数据 29, 则输出 “29 is a prime number”。

方法二:

还可以采用如下办法: 先定义一个变量 **flag**, 用它来表示 **m** 是否为素数, 可以假定 **flag** 的值为 1 时, 表示 **m** 是素数, **flag** 的值为 0 时, 表示 **m** 不是素数。这个变量 **flag** 通常被称为 “标志变量”, 在以后的学习中还会遇到这种变量。可以事先假定 **m** 是一个素数, 即把 **m** 赋初值为 1, 如果 **m** 能被 2 到 **k** 中任何一个整数整除, 那么就把 **flag** 的值置为 0。这样, 在循环结束时, 通过 **m** 的值就可以判断 **m** 是否为素数。程序如下:

```

#include<stdio.h>
#include<math.h>
int main()
{
    int m,i,k,flag;          /*定义标志变量*/
    scanf("%d",&m);
    k=sqrt(m);
    flag=1;                  /*假设 m 是素数*/
    for(i=2;i<=k;i++)
        if(m%i==0)
        {
            flag=0;          /*表示 m 不是素数*/
            break;           /*跳出循环*/
        }
    if(flag==1)
        printf("%d is a prime number\n",m);
    else
        printf("%d is not a prime number\n",m);
    return 0;
}

```

请思考：在方法二中，如果把下面的 if 语句：

```
if(flag==1) printf("%d is a prime number\n",m);
```

改为：

```
if(flag) printf("%d is a prime number\n",m);
```

会出现什么情况呢？

【案例 5-27】 编写程序，输出 100 以内所有的素数(每行输出 10 个数)。

案例分析：案例 5-26 学习了如何判断一个指定的数是否是素数，本案例中要求求出所有 100 以内的素数，问题可归结为：对 2~100 间的所有整数逐一判断是否是素数，可在案例 5-26 的基础上，添加一个外层循环，对 2~100 间的整数进行穷举处理。为控制输出格式，可增加一个计数变量 n，每判断出一个素数，令 n 加 1，当 n 是 10 的倍数时，输出换行。程序如下：

```
#include<stdio.h>
#include<math.h>
int main()
{
    int m,i,k;
    int n=0;                                /*n 是计数变量，统计个数，并可控制换行*/
    for(m=2;m<100;m++)
    {
        k=sqrt(m);
        for(i=2;i<=k;i++)
            if(m%i==0) break;              /*m 能被某个 i 整除，已不是素数，结束循环*/
        if(i>k)
        {
            printf("%5d",m);
            n++;
            if(n%10==0) printf("\n");      /*每输出 10 个后换行*/
        }
    }
    printf("\n100 以内共有%d 个素数。 \n",n);
    return 0;
}
```

程序运行结果如图 5-41 所示。

```

 2   3   5   7  11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73  79  83  89  97
100以内共有25个素数。
Press any key to continue
```

图 5-41 运行结果

算法效率改进：根据数学常识，除 2 以外，其余的素数一定是奇数，所以可以对 for 循环做修改，使循环次数减少一半，改进后的程序如下：

```
#include<stdio.h>
#include<math.h>
```

```

int main()
{
    int m,i,k;
    int n=1;
    printf("%5d",2);          /*首先输出素数2，循环从3开始只考察奇数*/
    for(m=3;m<1000;m=m+2)
    {
        k=sqrt(m);
        for(i=2;i<=k;i++)
            if(m%i==0) break;    /*m能被某个i整除，已不是素数，结束循环*/
        if(i>k)
        {
            printf("%5d",m);
            n++;
            if(n%10==0) printf("\n");
        }
    }
    printf("\n1000 以内共有%d 个素数。 \n",n);
    return 0;
}

```

107

程序执行情况相同，但效率提高一倍。注意分析两个程序中不同的地方，如 n 的初值、首个素数 2 的输出等关键点。

【案例 5-28】 编写程序，求 Fibonacci 数列前 30 项，每行输出 5 个数。

案例分析：

① 问题背景：Fibonacci 数列是中世纪意大利数学家在《算盘书》中提出的一个关于兔子繁殖的问题：如果一对兔子每月能生一对小兔，而每对小兔在它出生后的第三个月开始，每月又能生一对小兔，假定在不发生死亡的情况下，每个月有多少对兔子？

② 通过分析可以得出每个月兔子的对数应该是：

月份	1	2	3	4	5	6	7	...
兔子数	1	1	2	3	5	8	13	...

通过观察可以发现，每个月的兔子数量是有规律可循的，即：

第 i 个月兔子的对数=第 $(i-1)$ 个月兔子的对数+第 $(i-2)$ 个月兔子对数。

③ 算法设计思想：可以设 $f1$ 表示第 $(i-2)$ 个月兔子的对数， $f2$ 表示第 $(i-1)$ 个月兔子的对数， $f3$ 表示第 i 个月兔子的对数，即： $f3=f1+f2$ 。

1	1	2	3	5	8	13
$f1$	$f2$	$f3$				
	$f1$	$f2$	$f3$			
		$f1$	$f2$	$f3$		
			$f1$	$f2$	$f3$	
				$f1$	$f2$	$f3$

从以上数据可以看出，先从第 1 项开始， $f1$ 、 $f2$ 分别表示第 1 项和第 2 项，初值均为 1，第 3 项的值 $f3=f1+f2$ 。

计算第 4 项的值：这时的 f3 表示的是第 4 项，那么 f1 表示的就应该是第 2 项，即刚才的 f2，f2 表示的就应该是第 3 项，即刚才的 f3。因此，在使用公式 $f3=f1+f2$ 计算第 4 项的值之前，需要先把 f2 的值赋给 f1 ($f1 \leftarrow f2$)，把 f3 的值赋给 f2 ($f2 \leftarrow f3$)。然后，再使用公式计算，得出的值 f3 就是第 4 项的值。以此类推，就可以求出后面各项的值。

从上面的分析可知，从数列的第 3 项开始，每一项的值都依赖于其前两项，这种方法叫递推法。递推算法的基本思想是：从初值出发，归纳出新值与旧值间的关系，直到推出所需值为止。即新值的求出依赖于旧值，不知道旧值就无法推导出新值，类似于数学上的递推公式。

程序代码如下：

```
#include<stdio.h>
int main()
{
    int f1,f2,f3,i;
    f1=1;f2=1;
    printf("%10d%10d",f1,f2);
    for(i=3;i<=30;i++)          /*从第 3 项开始计算*/
    {
        f3=f1+f2;
        printf("%10d",f3);
        if(i%5==0) printf("\n");    /*每输出 5 个后换行*/
        f1=f2;
        f2=f3;
    }
    return 0;
}
```

程序运行结果如图 5-42 所示。

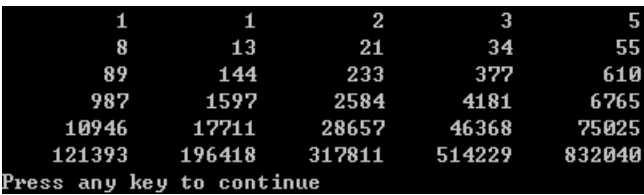


图 5-42 运行结果

【案例 5-29】 编写程序，输出如下所示图形。

```

D   D   D   D   D   D   D
    C   C   C   C   C
      B   B   B
        A
```

案例分析：此题属于图形输出的问题，因循环次数已知，这类问题可以采用双重 for 循环实现。外循环控制行的输出，内循环控制每行输出的字符。图形的每行可视为由行前导空格

和行内字符两部分构成，算法设计的核心是，探究行号与每行前导空格数及行内字符个数之间的对应关系。

分析本题图形规律如下。

行号：用 for 循环，行号 i 作为循环控制变量，取值 1、2、3、4。

前导空格：第 i 行 (1、2、3、4) 对应的空格数为 $2*i$ (2、4、6、8)。

每行字母：第 i 行 (1、2、3、4) 对应的字母个数为 $9-2*i$ (7、5、3、1)，每行输出的字符一样。

每行结束，输出换行控制，字符递减，可用字符变量值减 1 实现。

程序如下：

```
#include<stdio.h>
int main()
{
    int i,j;
    char ch='D';
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=i;j++)
            printf(" ");
        /*每次循环输出 2 个空格，共循环 i 次，产生 2*i 个空格*/
        for(j=1;j<=9-2*i;j++)
            printf("%c",ch);
        /*每次循环输出 1 个字母和 1 个空格，共循环 9-2*i 次，产生本行字符*/
        printf("\n");
        ch--;    /*字符变量 ch 减 1，为下一行字符输出做准备*/
    }
    return 0;
}
```

运行结果如图 5-43 所示。

```
D D D D D D D
  C C C C C
    B B B
      A
Press any key to continue
```

图 5-43 运行结果

【案例 5-30】 编写程序，输出如下所示图形。

```

      A
    A  B  C
  A  B  C  D  E
A  B  C  D  E  F  G
```

案例分析：这是一个正三角图形，图形规律如下。

行号：用 for 循环，行号 i 作为循环控制变量，取值 1、2、3、4。

前导空格：呈递减规律，每行递减 2 个空格。

每行字母：第 i 行 (1、2、3、4) 对应的字母个数为 $2*i-1$ (1、3、5、7)，每行字符从 A 开始，依次递增。

每行结束，输出换行控制。

根据分析，编写程序如下：

```
#include<stdio.h>
int main()
{
    int i,j;
    char ch;
    for(i=1;i<=4;i++)
    {
        ch='A';           /*控制每行字符从'A'开始*/
        for(j=1;j<=7-i;j++) /*空格随行递减，用 7-i 控制*/
            printf(" ");
        for(j=1;j<=2*i-1;j++)
        {
            printf("%c",ch);
            ch++;           /*行内字符递增*/
        }
        printf("\n");
    }
    return 0;
}
```

运行结果如图 5-44 所示。



```

      A
     A B C
    A B C D E
   A B C D E F G
Press any key to continue
```

图 5-44 运行结果

5.5 本章小结

本章首先讲解了算法的基本概念和程序的运行流程图，然后讲解了 C 语言中基本的流程控制语句：选择结构和循环结构。通过对本章案例的学习与实践，读者应能够熟练运用 if 选择语句、switch 选择语句、while 循环语句、do...while 循环语句及 for 循环语句。在难度稍高的案例中，进一步应用了三种循环语句和选择语句的相互嵌套，处理复杂的问题。读者应尽可能地掌握本章所有案例内容，并能将案例中的知识应用于实践，为后续章节的学习打下坚实的基础。

5.6 习题

一、单选题

- 已有定义“int x=3,y=4,z=5;”,则表达式“!(x+y)+z-1 && y+z/2”的值是()。
A. 6 B. 0 C. 2 D. 1
- 设 a=5,b=6,c=7,d=8,m=2,n=2,则执行“(m=a>b) && (n=c>d)”后 n 的值为()。
A. 1 B. 2 C. 3 D. 4
- 设 x、y 和 z 都是 int 类型变量,且 x=3,y=4,z=5,则下面的表达式中,值为 0 的表达式为()。
A. 'x' && 'y' B. x<=y C. x||y+z && y-z D. !((x<y) && !z||1)
- 为了避免嵌套的 if...else 语句的二义性,C 语言规定 else 总是与()组成配对关系。
A. 缩排位置相同的 if B. 在其之前未配对的 if
C. 在其之前未配对的最近的 if D. 同一行上的 if
- 逻辑运算符两侧运算对象的数据类型()。
A. 只能是 0 或 1 B. 只能是 0 或非 0 正数
C. 只能是整型或字符型数据 D. 可以是任何类型的数据
- 以下关于运算符优先顺序的描述中正确的是()。
A. 关系运算符<算术运算符<赋值运算符<逻辑与运算符
B. 逻辑与运算符<关系运算符<算术运算符<赋值运算符
C. 赋值运算符<逻辑与运算符<关系运算符<算术运算符
D. 算术运算符<关系运算符<赋值运算符<逻辑与运算符
- 下列运算符中优先级最高的是()。
A. < B. && C. + D. !=
- 以下合法的 if 语句是(设 int x,a,b,c;)()。
A. if(a=b) c++; B. if(a=<b) c++;
C. if(a<>b) c++; D. if(a=>b) c++;
- 能正确表示“当 x 的取值在[-58,-40]和[40,58]范围内为真,否则为假”的表达式是()。
A. (x>=-58) && (x<=-40) && (x>=40) && (x<=58)
B. (x>=-58) || (x<=-40) || (x>=40) || (x<=58)
C. (x>=-58) && (x<=-40) || (x>=40) && (x<=58)
D. (x>=-58) || (x<=-40) && (x>=40) || (x<=58)
- 判断 char 型变量 s 是否为小写字母的正确表达式是()。
A. 'a' <= s <='z' B. (s>='a') & (s<='z')
C. (s>='a') && (s<='z') D. ('a'<=s) and ('z'>=s)
- 若希望当 x 的值为奇数时,表达式的值为“真”,x 的值为偶数时,表达式的值为“假”,

则以下不能满足要求的表达式是()。

- A. $x\%2==1$ B. $!(x\%2==0)$
C. $!(x\%2)$ D. $x\%2$

12. 已知 “ $x=45, y='a', z=0;$ ”, 则表达式 $(x>=z \ \&\& \ y<'z' \ || \ !y)$ 的值是()。

- A. 0 B. 语法错 C. 1 D. “假”

13. 指出下列程序段表示的逻辑关系是()。

```
if(a<b)
{
    if(c==d)
        x=10;
}
else
    x=-10;
```

- A. $x = \begin{cases} 10 & a < b \text{ 且 } c = d \\ -10 & a \geq b \text{ 且 } c \neq d \end{cases}$ B. $x = \begin{cases} 10 & a < b \text{ 且 } c = d \\ -10 & a \geq b \end{cases}$
C. $x = \begin{cases} 10 & a < b \text{ 且 } c = d \\ -10 & a < b \text{ 且 } c \neq d \end{cases}$ D. $x = \begin{cases} 10 & a < b \text{ 且 } c = d \\ -10 & c \neq d \end{cases}$

14. 有函数: $y = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$, 则以下程序段中不能根据 x 值正确计算出 y 值的是()。

- A. `if (x>0) y=1;`
 `else if (x==0) y=0;`
 `else y=-1;`
B. `y=0;`
 `if (x>0) y=1;`
 `else f (x<0) y=-1;`
C. `y=0;`
 `if (x>=0)`
 `if(x>0) y=1;`
 `else y=-1;`
D. `if (x>=0)`
 `if (x>0) y=1;`
 `else y=0;`
 `else y=-1;`

15. 执行以下语句后, y 的值为()。

```
int x,y,z;
x=y=z=0;
++x || ++y && ++z;
```

- A. 0 B. 1 C. 2 D. 不确定值

16. 已知 “`int a=1,b=2,c=3;`”, 以下语句执行后 a,b,c 的值是()。

```
if(a>b)
c=a; a=b; b=c;
```

- A. $a=1, b=2, c=3$ B. $a=2, b=3, c=3$
C. $a=2, b=3, c=1$ D. $a=2, b=3, c=2$

17. 请阅读以下程序, 该程序()。

```
#include "stdio. h"
main()
{   int x=-10, y=5, z=0;
    if (x=y+z) printf("***\n");
    else printf("$$$\n");
}
```

- A. 有语法错, 不能通过编译 B. 可以通过编译, 但不能通过链接
C. 输出*** D. 输出\$\$\$

18. 以下程序的运行结果是()。

```
#include "stdio.h"
main()
{   int a=1;
    if (a++>1) printf("%d\n",a);
    else printf("%d\n",a--);
}
```

- A. 0 B. 1 C. 2 D. 3

19. 当 a=1,b=2,c=4,d=3 时, 执行完下面一段程序后 x 的值是()。

```
if (a<b)
if (c<d) x=1;
else
    if (a<c)
        if (b<d) x=2;
        else x=3;
    else x=4;
else x=5;
```

- A. 1 B. 2 C. 3 D. 4

20. 执行以下程序段后, 变量 x,y,z 的值分别为()。

```
int a=1,b=0,x,y,z;
x=(--a==b++)? -a:++b;
y=a++;
z=b;
```

- A. x=0,y=0,z=0 B. x=-1,y=-1,z=1
C. x=0,y=1,z=0 D. x=-1,y=2,z=1

21. 若 a,b,c,d,w 均为 int 类型变量, 则执行下面语句后 w 的值是()。

```
a=1;b=2;c=3;d=4;
w=(a<b)? a:b;
w=(w<c)? w:c;
w=(w<d)? w:d;
```

- A. 1 B. 2 C. 3 D. 4

22. 以下程序的输出结果是()。

```
#include "stdio.h"
```

```
main()
{
    int a=5,b=4,c=6,d;
    printf("%d\n",d=a>b?a>c?a:c:b);
}
```

A. 5 B. 4 C. 6 D. 不确定

23. 若 a,b,c1,c2,x,y 均为整型变量, 正确的 switch 语句是()。

A. switch (a+b);

B. switch a

```
{
    case 1: y=a+b; break;
    case 0: y=a-b;
        break;
}
```

```
{
    case c1: y=a-b; break;
    case c2: x=a*d; break;
    default: x=a+b;
}
```

C. switch (a*a+b*b)

D. switch (a-b)

```
{
    case 3:
    case 1: y=a+b; break;
    case 3: y=b-a; break;
}
```

```
{
    default: y=a*b; break;
    case 3: x=a+b; break;
    case 10: case 11: y=a-b; break;
}
```

24. 执行下列程序, 输入 3, 输出结果是()。

```
#include "stdio.h"
main()
{
    int k;
    scanf("%d",&k);
    switch(k)
    { case 1: printf("%d\n",k++);
      case 2: printf("%d\n",k++);
      case 3: printf("%d\n",k++);
      case 4: printf("%d\n",k++);
      break;
      default: printf("Full!\n");
    }
}
```

A. 3

B. 4

C. 3

D. 4

5

4

25. 假定等级和分数有以下对应关系:

等级 A, 分数 85~100;

等级 B, 分数 60~84;

等级 C, 分数 60 以下。

有等级 grade, 输出相应的分数区间, 能够完成该功能的程序段是()。

A. switch (grade)
{ case 'A': printf("85~100\n");
case 'B': printf("60~84\n");
case 'C': printf("<60\n");
default: printf("grade is error!\n");
}

B. switch (grade)
{ case 'A': printf("85~100\n");
break;
case 'B': printf("60~84\n");
case 'C': printf("<60\n");
default: printf("grade is error!\n");
}

C. switch (grade)
{ case 'A': printf("85~100\n");
break;
case 'B': printf("60~84\n");
break;
case 'C': printf("<60\n");
default: printf("grade is error!\n");
}

D. switch (grade)
{ case 'C': printf("<60\n");
break;
case 'B': printf("60~84\n");
break;
default: printf("grade is error!\n");
break;
case 'A': printf("85~100\n");
}

26. 以下程序的运行结果是()。

```
#include "stdio.h"
main()
{
    float x=2.0,y;
    if(x<0.0)y=0.0;
    else if(x<10.0)y=1.0/x;
        else y=1.0;
    printf("%f\n",y);
}
```

A. 0.000000 B. 0.250000 C. 0.500000 D. 1.000000

27. 以下程序的运行结果是()。

```
#include "stdio.h"
main()
{
    int a=2,b=-1,c=2;
    if (a<b)
        if (b<0)
            c=0;
        else c++;
    printf("%d\n",c);
}
```

A. 0 B. 1 C. 2 D. 3

28. 运行以下程序后，从键盘输入 china#，则输出为()。

```
#include "stdio.h"
main()
{
    int v1=0,v2=0;
    char ch;
    while ((ch=getchar())!='#')
        switch (ch)
        { case 'a':
          case 'h':
            default: v1++;
          case 'o':v2++;
        }
    printf("%d,%d\n",v1,v2);
}
```

A. 2,0 B. 5,0 C. 5,5 D. 2,5

29. 有以下程序，若输入为字符 s，则程序运行结果为()。

```
#include "stdio.h"
main()
{
    char ch;
    ch=getchar();
    switch (ch)
    { case 'a':printf("a=%c\n",ch);
      default: printf("end!\n");
      case 'b':printf("b=%c\n",ch);
      case 'c':printf("c=%c\n",ch);
    }
}
```

[illegible]

30. 有以下程序，程序运行后的输出结果是()。

```
#include "stdio.h"

main()
{
    int a=15, b=21, m=0;
    switch (a%3)
    {
        case 0: m++; break;
        case 1: m++;
        switch (b%2)
        {
            default: m++;
            case 0: m++; break;
        }
    }

    printf("%d\n",m);
}
```

A. 1 B. 2 C. 3 D. 4

31. 以下程序的输出结果是()。

```
main()
{   int n=4;
    while (n--)
        printf("%d",--n);
}
```

A. 20 B. 31 C. 321 D. 210

32. 以下程序的输出结果是()。

```
main()
{  int x=10,y=10,i;
    for(i=0;x>8;y++i)
        printf("%d%d",x--,y);
}
```

A. 10192 B. 9876 C. 10990 D. 101091

33. 当执行以下程序段时()。

```
x=-1;
do
{ x=x*x; }
while (!x);
```

A. 循环体将执行一次 B. 循环体将执行两次
C. 循环体将执行无数次 D. 系统将提示有语法错误

34. 执行以下程序后输出的结果是()。

```
main()
{
    int y=10;
    do{y--;}while(--y);
    printf("%d\n",y--);
}
```

- A. -1 B. 1 C. 8 D. 0

35. 以下程序的输出结果是()。

```
main()
{
    int x=3,y=6,a=0;
    while (x++!=(y-=1))
    {
        a+=1;
        if (y<x) break;
    }
    printf("x=%d,y=%d,a=%d\n",x,y,a);
}
```

- A. x=4,y=4,a=1 B. x=5,y=5,a=1
C. x=5,y=4,a=3 D. x=5,y=4,a=1

36. 若 i 和 j 已定义为 int 类型, 则以下程序段中内循环的总的执行次数是()。

```
for (i=5;i;i--)  
for (j=0;j<4;j++)  
{...}
```

- A. 20 B. 24 C. 25 D. 30

37. 以下程序的输出结果是()。

```
#include<stdio.h>  
main()  
{  
    int x=1,y=0,a=0,b=0;  
    switch(x)  
    {  
        case 1:  
            switch(y)  
            {  
                case 0:a++;break;  
                case 1:b++;break;  
            }  
            case 2:a++;b++;break;  
    }  
    printf("a=%d,b=%d\n",a,b);  
}
```

- A. a=2,b=1 B. a=1,b=1 C. a=1,b=0 D. a=2,b=2

38. 若有以下程序段, w 和 k 都是整型变量, 则不能与程序段等价的循环语句是()。

```
w=k;
LB: if(w==0) goto LE;
    w--;
    printf("*");
    goto LB;
LE:
```

- A. `for(w=k;w!=0;w--)`
 `print("*");`
- B. `w=k;`
 `while(w--!=0)`
 `printf("*");`
 `w++;`
- C. `w=k;`
 `do{w--;printf("*");}`
 `while (w!=0);`
- D. `for (w=k;w;--w) printf("*");`

39. 以下程序的输出是()。

```
main()
{
    char *s="12134211";
    int k,v1=0,v2=0,v3=0,v4=0;
    for (k=0;s[k];k++)
        switch(s[k])
        {
            default:v4++;
            case 1:v1++;
            case 2:v2++;
            case 3:v3++;
        }
    printf("v1=%d,v2=%d,v3=%d,v4=%d\n",v1,v2,v3,v4);
}
```

- A. $v1=4, v2=2, v3=1, v4=1$
- B. $v1=4, v2=9, v3=3, v4=1$
- C. $v1=5, v2=8, v3=6, v4=1$
- D. $v1=8, v2=8, v3=8, v4=8$

40. 以下程序的输出结果是()。

```
#include"stdio.h"
int abc(int u,int v );
main()
{
    int c, a=24, b=16;
    c=abc(a,b);
    printf("%d\n",c);
}
```

```

    }
    int abc (int u,int v)
    {
        int w;
        while (v)
            {w=u%v;u=v;v=w;}
        return u;
    }

```

A. 6 B. 7 C. 8 D. 9

41. 在下列选项中, 没有构成死循环的程序段是()。

- A. `int i=100;`
`while (1)`
`{`
`i=i%100+1;`
`if(i>100) break;`
`}`
- B. `for (;);`
- C. `int k=1000;`
`do {++k;}while (k>=10000);`
- D. `int s=36;`
`while (s);`
`--s;`

42. 若输入 B, 以下程序的运行结果为()。

```

main()
{
    char grade;scanf("%c",&grade);
    switch(grade)
    {
        case'A':printf(">=85.");
        case'B':
        case'C':printf(">=60.");
        case'D':printf("<60.");
        default:printf("error.");
    }
}

```

A. `>=85.` B. `>=60.` C. `>=60.<60.error.` D. `error.`

43. 以下程序的运行情况是()。

```

main()
{
    int i=1,sum=0;
    while(i<10)sum=sum+1;i++;
    printf("i=%d,sum=%d",i,sum);
}

```

- A. $i=10, \text{sum}=9$ B. $i=9, \text{sum}=9$
 C. $i=2, \text{sum}=1$ D. 运行出现错误
44. 与以下程序语句不等价的是()。

```
i=1;
for(;i<=100;i++) sum+=i;
```

- A. `for(i=1; ;i++) {sum+=i;if(i==100) break;}`
 B. `for(i=1;i<=100;) {sum+=i;i++;}`
 C. `i=1;for(;i<=100;) sum+=i;`
 D. `i=1;for(;) {sum+=i;if(i==100) break;i++;}`
45. 以下程序的运行结果是()。

```
main()
{
    int n;
    for(n=1;n<=10;n++)
    {
        if(n%3==0) continue;
        printf("%d", n);
    }
}
```

- A. 12457810 B. 369 C. 12 D. 12345678910
46. 以下程序的运行结果是()。

```
main()
{
    int x,y,z;
    x=0;y=z=-1;
    x+=-z---y;
    printf("x=%d\n",x);
}
```

- A. $x=4$ B. $x=0$ C. $x=2$ D. $x=3$
47. 标有`/**/`的语句的执行次数是()。

```
int y,i;
for(i=0;i<20;i++)
{
    if(i%2==0) continue;
    y+=i; /**/
}
```

- A. 20 B. 19 C. 10 D. 9
48. 在 C 语言中, `if` 语句后的一对圆括号中, 用来决定分支流程的表达式()。
- A. 只能用逻辑表达式
 B. 只能用关系表达式

C. 只能用逻辑表达式或关系表达式

D. 可用任意表达式

49. 在以下给出的表达式中, 与 `do...while(E)` 语句中的 (E) 不等价的表达式是()。

A. `(!E==0)` B. `(E>0||E<0)` C. `(E==0)` D. `(E!=0)`

50. 假设所有变量均已正确定义, 下列程序段运行后 x 的值是()。

```
k1=1;k2=2; k3=3; x=15;
if(!k1)x--;
else
    if(k2)
        if(k3)x=4;
        else x=3;
```

A. 14

B. 4

C. 15

D. 3

51. 执行以下语句的输出是()。

```
int i=-1;
if(i<=0)printf("****\n");
else printf("%%%\n");
```

A. ****

B. % % % %

C. % % % % c

D. 有错, 执行不正确

52. 以下程序的输出是()。

```
#include<stdio.h>
main()
{
    int i;char c;
    for(i=0;i<=5;i++)
    {
        c=getchar();putchar(c);
    }
}
```

程序执行时从第一列开始输入以下数据, <CR>代表换行符。

```
u<CR>
w<CR>
xsta<CR>
```

A. uwxsta

B. u

C. u

D. u

w

w

w

x

xs

xsta

53. 以下程序的输出是()。

```
#include"stdio.h"
main()
{
    int i,j,x=0;
    for(i=0;i<2;i++)
    {
```

```

        x++;
        for(j=0;j<=3;j++)
        {if(j%2)continue;x++;}
        x++;
    }
    printf("x=%d\n",x);
}

```

- A. x=4 B. x=8 C. x=6 D. x=12

54. 以下程序的输出是()。

```

#include<stdio.h>
main()
{
    int i,j,k=0,m=0;
    for(i=0;i<2;i++)
    { for(j=0;j<3;j++)k++;k-=j;}
    m=i+j;
    printf("k=%d,m=%d\n",k,m);
}

```

- A. k=0,m=3 B. k=0,m=5 C. k=1,m=3 D. k=1,m=5

55. 在 C 语言中, 为了结束 while 语句构成的循环, while 后一对圆括号中表达式的值应该为()。

- A. 0 B. 1 C. true D. 非 0

56. 在 C 语言中, 为了结束由 do...while 语句构成的循环, while 后一对圆括号中表达式的值应为()。

- A. 0 B. 1 C. true D. 非 0

57. 以下程序的输出是()。

```

#include<stdio.h>
main()
{
    int k=0;char c='A';
    do
    {
        switch(c++)
        {
            case 'A':k++;break;
            case 'B':k--;
            case 'C':k+=2;break;
            case 'D':k=k%2;continue;
            case 'E':k=k*10;break;
            default:k=k/3;
        }
        k++;
    }while(c<'G');
}

```

```
printf("%d\n", k);
}
```

- A. k=3 B. k=4 C. k=2 D. k=0

58. 以下程序的输出是()。

```
#include<stdio.h>
main()
{
    int i=0,j=0,a=6;
    if((++i>0)||(++j>0))a++;
    printf("i=%d,j=%d,a=%d\n",i,j,a);
}
```

- A. i=0,j=0,a=6 B. i=1,j=1,a=7 C. i=1,j=0,a=7 D. i=0,j=1,a=7

59. 以下程序的输出是()。

```
main()
{
    int x=3;
    do
    {
        printf("%3d",x--=2);
    }
    while(!(--x));
}
```

- A. 1 B. 3 0 C. 1 -2 D. 死循环

二、填空题

1. 执行以下程序，若从键盘输入 58，则输出结果是_____。

```
#include "stdio.h"
main()
{
    int a;
    scanf("%d",&a);
    if (a>50) printf("%d",a);
    if (a>40) printf("%d",a);
    if (a>30) printf("%d",a);
}
```

2. 设 “int x=9,y=8;”，表达式 “x==y+1” 的结果是_____。

3. 定义 “int x,y;”，执行 “y=(x=1,++x,x+2);” 后 y 的值是_____。

4. 定义 “int x=10,y,z;”，执行 “y=z=x; x=y==z;” 后 x 的值是_____。

5. 设 “int a=1,b=2,c,d,e;”，执行 “c=(-a++)+(++b); d=(b--)+(++a)-a; e=(a/(++b))-(a/(-a));” 后 a,b,c,d,e 的值是_____。

6. 设 “int a=2,b=3,c,d,e,f;”，执行 “c=(a++>==b); d=(a==++b); e=(a--!=b); f=(++a>b--);” 后 a,b,c,d,e,f 的值是_____。

7. 以下程序的运行结果是_____。

```
#include "stdio.h"
main()
{
    int a,b,c,s,w,t;
    s=w=t=0;
    a=-1; b=3; c=3;
    if (c>0) s=a+b;
    if (a<=0)
    { if (b>0)
        if (c<=0) w=a-b;
    }
    else if (c>0) w=a-b;
        else t=c;
    printf("%d%d%d",s,w,t);
}
```

8. 以下程序的运行结果是_____。

```
#include "stdio.h"
main()
{
    int a,b,c,d,e;
    a=c=1;
    b=20;
    d=100;
    if(!a)d=d++;
    else if(!b)
        if(d)d=--d;
    else d=d--;
    printf("%d\n\n",d);
}
```

9. 以下程序的运行结果是_____。

```
#include "stdio.h"
main()
{
    int a,b= 250,c;
    if ((c=b)<0) a=4;
    else if (b=0) a=5;
    else a=6;
    printf("\t%d\t%d\n",a,c);
    if (c=(b==0))
        a=5;
    printf("\t%d\t%d\n",a,c);
    if (a=c=b) a=4;
    printf("\t%d\t%d\n",a,c);
}
```

10. 下面程序根据以下函数关系, 对输入每个 x 值, 计算出 y 值。填空实现程序功能。

x	y
$2 < x \leq 10$	$x(x+2)$
$-1 < x \leq 2$	$1/x$
$x \leq -1$	$x-1$

```
#include "stdio.h"
main()
{
    int x,y;
    scanf("%d",&x);
    if (_____) y=x*(x+2);
    else if (_____) y=1/x;
        else if (x<=-1) y=x-1;
            else _____;
    if (y!=-1) printf("%d",y);
    else printf("error");
}
```

126

11. 以下程序的功能是计算一元二次方程 $ax^2+bx+c=0$ 的根。填空实现程序功能。

```
#include "stdio.h"
#include "math.h"
main()
{
    float a,b,c,t,disc,w,term1,term2;
    printf("enter a,b,c:");
    scanf("%f%f%f",&a,&b,&c);
    if (_____)
        if (_____)
            printf("no answer due to input error\n");
        else
            printf("the single root is %f\n",-c/b);
    else
    {
        disc=b*b-4*a*c;
        w=2*a;
        term1=-b/w;
        t=abs(disc);
        term2=sqrt(t)/w;
        if (_____)
            printf("complex root\n real part=%f imag part =%f\n", term1,term2);
        else
            printf("real roots\n root1=%f root2=%f\n", term1+term2,term1-term2);
    }
}
```


12. 以下程序根据输入的三角形的三边判断是否能组成三角形, 若可以组成三角形, 则输出三角形的面积和三角形的类型。填空实现程序功能。

```
#include "math.h"
#include "stdio.h"
main()
{
    float a,b,c,s,area;
    printf("please input three edges of a triangle:");
    scanf("%f%f%f",&a,&b,&c);
    if (_____)
    {
        s=(a+b+c)/2;
        area=sqrt(s*(s-A*(s-B*(s-c)));
        printf("\nthe area of the triangle is: %f",area);
        if ((a==b)&&(b==c))
            printf("等边三角形");
        else if (_____)
            printf("等腰三角形");
        else if (_____)
            printf("直角三角形");
        else printf("一般三角形");
    }
    else printf("不能组成三角形");
}
```

13. 以下程序的功能是判断输入的年份是否是闰年。填空实现程序功能。

```
#include "stdio.h"
main()
{
    int year, flag;
    printf("please input the year to jude whether it is a leap year:");
    scanf("%d",&year);
    if (year%400==0) flag=1;
    else if (_____) flag=1;
    else _____;
    if (flag) printf("%d is a leap year\n",year);
    else printf("%d is not a leap year!\n",year);
}
```

14. 以下程序是对用户输入的字母进行大小写转换。填空实现程序功能。

```
#include "stdio.h"
main()
{
    char ch;
    printf("please input a letter:");
    scanf("%c",&ch);
    if (_____) ch=ch+32;
```

```

else if (ch>='a' && ch<='z')
    _____;
printf("the converted letter is: %c\n",ch);
}

```

15. 以下程序是对从键盘输入的任何三个整数，求出其中的最小值。填空实现程序功能。

```

#include "stdio.h"
main()
{
    int a,b,c,min;
    printf("please input three numbers:");
    scanf("%d%d%d",&a,&b,&c);
    if (_____)
        min=b;
    else
        min=a;
    if (min>c)
        _____;
    printf("min=%d\n",min);
}

```

128

16. 以下程序实现的功能是：商店卖西瓜，10 斤以上的每斤 0.15 元(含 10 斤)，8 斤以上的每斤 0.3 元(含 8 斤)，6 斤以上的每斤 0.4 元(含 6 斤)，4 斤以上的每斤 0.6 元(含 4 斤)，4 斤以下的每斤 0.8 元，从键盘输入西瓜的重量和顾客所付钱数，则输出应付款和应找钱数。填空实现程序功能。

```

#include "stdio.h"
main()
{
    float weight, money, rate;
    printf("the paid money of the client is:");
    scanf("%f",&money);
    printf("the weight of the watermelon is:");
    scanf("%f",&weight);
    if (_____)
        rate=0.15;
    else if (weight>=8)
        rate=0.3;
    else if (weight>=6)
        _____;
    else if (weight>=4)
        rate=0.6;
    _____
        rate=0.8;
    printf("the account payable of the watermelon is %f\n", weight*rate);
    printf("the change for client is %f\n",money-weight*rate);
}

```

17. 以下程序的运行结果是_____。

```
#include "stdio.h"
main()
{
    char ch1='a',ch2='A';
    switch (ch1)
    { case 'a':
      switch (ch2)
      { case 'A': printf("good!\n"); break;
        case 'B': printf("bad!\n"); break;
      }
      case 'b': printf("joke\n");
    }
}
```

18. 以下程序完成两个数的四则运算。填空实现程序功能。

```
#include "stdio.h"
main()
{
    float x,y;
    char operator;
    printf("please input the expression:");
    scanf("%f%c%f",&x,&operator,&y);
    switch (_____)
    { case '+': printf("%g%c%g=%g\n",x,operator,y,x+y);
      _____;
      case '-': printf("%g%c%g=%g\n",x,operator,y,x-y);
        break;
      case '*': printf("%g%c%g=%g\n",x,operator,y,x*y);
        break;
      case '/': if (y==0.0)
        printf("除零错误! \n");
        else
        printf("%g%c%g=%g\n",x,operator,y,x/y);
        break;
      _____: printf("表达式存在错误!\n");
    }
}
```

19. 以下程序运行后的输出结果是_____。

```
#include "stdio.h"
main()
{
    int x=10,y=20,t=0;
    if(x==y)t=x;x=y;y=t;
    printf("%d,%d\n",x,y);
}
```

20. 执行以下语句后, x、y 和 z 的值分别为_____。

```
int x,y,z;  
x=y=z=0;  
++x || ++y && ++z;
```

21. 若输入字母 C, 程序的输出结果为_____; 若输入字符*, 程序的输出结果为_____。

```
#include<stdio.h>  
main()  
{  
    char c1,c2;  
    c1=getchar();  
    while(c1<97||c1>122)c1=getchar();  
    c2=c1-32;  
    printf("%c,%c\n",c1,c2);  
}
```

22. 以下程序的运行结果是_____。

```
#include<stdio.h>  
#include<math.h>  
main()  
{  
    int i,k,m,n=0;  
    for(m=1;m<=10;m+=2)  
    {  
        if(n%10==0)printf("\n");  
        k=sqrt(m);  
        for(i=2;i<=k;i++)  
            if(m%i==0)  
                break;  
        if(i>k)  
        {  
            printf("%2d",m);  
            n++;  
        }  
    }  
}
```

23. 如果输入 1、2、3、4, 程序运行输出的是_____。

```
#include<stdio.h>  
main()  
{ char c;  
    int i,k;  
    k=0;  
    for(i=0;i<4;i++)  
    {while(1)
```

```

        { c=getchar();if(c>='0'&&c<='9')break;}
        k=k*10+c-'0';
    }
    printf("k=%d\n",k);
}

```

24. 运行以下程序后, 如果从键盘上输入 china#<回车>, 则输出结果为_____。

```

#include<stdio.h>
main()
{  int  v1=0,v2=0;
   char ch;
   while ((ch=getchar())!='#')
       switch(ch)
       {  case 'a':
          case 'h':
          default: v1++;
          case '0':v2++;
        }
   printf("%d,%d\n",v1,v2);
}

```

25. 以下程序的运行结果是_____。

```

#include<stdio.h>
main()
{int i;
  for(i=1;i+1;i++)
  {  if(i>4){printf("%d\t",i++);break;}
    printf("%d\t",i++);
  }
}

```

26. 以下程序的运行结果是_____。

```

#include<stdio.h>
main()
{  int a,b;
   for(a=1,b=1;a<=100;a++)
   {  if(b>=20)break;
      if(b%3==1)
      {  b+=3;
         continue;
      }
      b-=5;
   }
   printf("%d\n",a);
}

```

27. 以下程序的运行结果是_____。

```
main()
{   int i=1;
    while(i<10)
        if(++i%3!=1)continue;
        else printf("%d",i);
}
```

28. 以下程序的运行结果是_____。

```
main()
{   int n=0;
    while(n++<=1)
        printf("%d\t",n);
        printf("%d\n",n);
}
```

29. 以下程序运行时，输入“qwert?”，程序的运行结果是_____。

```
#include<stdio.h>
main()
{   char c;
    while((c=getchar())!='?')putchar(++c);
}
```

30. 以下程序：

```
#include<stdio.h>
main()
{   int m,n;
    scanf("%d%d",&m,&n);
    while(m!=n)
    {   while(m>n)m-=n;
        while(n>m)n-=m;
    }
    printf("m=%d\n",m);
}
```

当输入 65 和 14 时，运行结果是_____。

当输入 14 和 63 时，运行结果是_____。

当输入 25 和 125 时，运行结果是_____。

31. 以下程序实现求 1000 以内的“完全数”（提示：如果一个数恰好等于它的因子之和，则称该数为“完全数”。因子包括 1，不包括数本身。如 6 的因子是 1、2、3， $6=1+2+3$ ，则 6 是个“完全数”）。填空实现程序功能。

```
main()
{   int i,a,m;
    for(i=1;i<1000;i++)
    {   for(m=0,a=1;a<=i/2;a++)
        if(!(i%a)) _____;
        if(_____)printf("%4d",i);
    }
```

32. 假设 100 元可买 100 只玩具鸡, 其中公鸡 1 只 5 元, 母鸡 1 只 3 元, 小鸡 1 元 3 只, 求 100 元能买公鸡、母鸡、小鸡各多少只。填空实现程序功能。

```
#include<stdio.h>
main()
{  int  cocks,hens,chicks;
    cocks=0;
    while(cocks<=19)
    {  hens=0;
        while(hens<=33)
        {  chicks=100.0-cocks-hens;
            if(5.0*cocks+3.0*hens+chicks/3.0==100.0)
                printf("%d,%d,%d\n",cocks,hens,chicks);
            _____;
        }
        _____;
    }
}
```

33. 编写程序, 计算 Fibonacci 数列(1,1,2,3,5,8,13…)的前 40 项。填空实现程序功能。

```
main()
{  int i;long f1,f2;
    _____;
    for(i=0;i<20;i++)
    {  printf("%12ld%12ld",f1,f2);
        if(i%2)printf("\n");
        f1+=f2;
        _____;
    }
}
```

34. 求 100~499 之间的所有水仙花数, 即各位数字的立方和恰好等于该数本身的数。填空实现程序功能。

```
main()
{  int I,j,k,m,n;
    for(I=1;_____;I++)
        for(j=0;j<=9;j++)
            for(k=0;k<=9;k++)
            {_____;
                n=I*I*I+j*j*j+k*k*k;
                if(_____)
                    printf("%d",m);
            }
}
```

35. 程序输入两个整数 m 和 n , 求其最大公约数。填空实现程序功能。

```
main()
{   int a,b,num1,num2,temp;
    scanf("%d,%d",&num1,&num2);
    if(_____)
    { temp=num1;
      num1=num2;
      num2=temp;
    }
    a=num1;b=num2;
    while(b!=0)
    { temp=_____;
      a=b;
      b=temp;
    }
    printf("%d,%d",a,num1*num2/a);
}
```

134

36. 以下程序的运行结果是_____。

```
#include<stdio.h>
void main(void)
{
    int i,j=4;
    for(i=j;i<=2*j;i++)
        switch (i/j)
        {
            case 0 :
            case 1: printf("*"); break;
            case 2: printf("#"); break;
        }
}
```

37. 以下程序的运行结果是_____。

```
#include<stdio.h>
void main(void)
{
    int a=0,i;
    for(i=0;i<5;i++)
    {
        switch(i)
        {
            case 0:break;
            case 1:a+=2;
            case 2:a+=3;
            default:a+=4;
        }
    }
}
```



```
    }  
    printf("a=%d\n",a);  
}
```

38. 以下程序的运行结果是_____。

```
#include<stdio.h>  
void main(void)  
{  
    int x=2;  
    do{  
        printf("%d\t",x=x-2);  
    }while(!x);  
}
```

39. 以下程序的运行结果是_____。

```
#include<stdio.h>  
void main(void)  
{  
    int k=5,n=0;  
    while(k>0)  
    {  
        switch(k)  
        {  
            case 1:  
            case 3:n+=1;k--;break;  
            default:n=0;k--;  
            case 2:  
            case 4:n+=2;k--;break;  
        }  
    }  
    printf("%3d\n",n);  
}
```

40. 以下程序的运行结果是_____。

```
#include<stdio.h>  
void main()  
{  
    int i;  
    for(i=1;i<=5;i++)  
    {  
        if(i%2)  
            putchar('<');  
        else  
            continue;  
        putchar('>');  
    }  
}
```

```
    putchar('#');
}
```

三、上机操作题

1. 编写程序, 实现输入一个整数, 判断该数为奇数还是偶数。
2. 编写程序, 实现判断键盘上输入字符的种类(控制字符、大写字母、小写字母、数字或其他)。
3. 编写程序, 实现从键盘上输入三边长 a 、 b 、 c , 判断这三边能否组成一个三角形, 若能, 计算并输出三角形的面积(画出流程图)。提示: 构成三角形的条件是, 任意两边之和大于第三边; 三角形面积的计算公式是 $\text{area}=\sqrt{d \times (d-a) \times (d-b) \times (d-c)}$, 其中 $d=(a+b+c)/2$ 。
4. 编写程序, 实现输入一位学生的出生年月日, 并输入当前的年月日, 计算输出该学生的实际年龄。
5. 编写程序, 实现输入今天是星期几(1~7), 计算输出 90 日后是星期几。
6. 编写程序, 实现给定一个不多于 3 位的正整数, 判断它是几位数, 并分别输出它的每一位数字(画出流程图)。
7. 某商场进行打折促销活动, 消费金额 p 越高, 折扣 d 越大, 其标准如下:

$p < 200$	$d = 0\%$
$200 \leq p < 400$	$d = 5\%$
$400 \leq p < 600$	$d = 10\%$
$600 \leq p < 1000$	$d = 15\%$
$1000 \leq p$	$d = 20\%$

要求使用 switch 语句编程, 输入消费金额, 求其实际消费金额。

8. 编写程序, 使用 switch 语句, 实现输入一个百分制的成绩, 将其转换并输出等级, 5 个等级分别为: 90 分及以上为 A, 80~89 分为 B, 70~79 分为 C, 60~69 分为 D, 0~59 分为 E, 如输入 75, 则显示 C。
9. 编写程序, 求 100~2000 之间所有 3 的倍数的数之和, 当和大于 1000 时结束(画出流程图)。
10. 编写程序, 计算并输出以下数列前 20 项的和。要求结果保留 4 位小数。

$$2/1, 3/2, 5/3, 8/5, 13/8, 21/13 \cdots$$
11. 编写程序, 求 $\sum_{n=1}^{20} n!$ (即求 $1!+2!+3!+4!+\cdots+20!$ 的结果)。
12. 编写程序, 求 e 的近似值, 直到最后一项的绝对值小于 10^{-5} 时为止, 输出保留 5 位小数。

$$e = \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

13. 编写程序, 统计并逐行显示(每行 5 个数)区间[10000,50000]中的回文数。回文数的含义是: 从左向右读与从右向左读是相同的, 即对称, 如 12321。
14. 编写程序, 输出所有三位数中的素数(每行输出 5 个数, 并画出流程图)。
15. 编写程序, 使用双重循环输出以下图形。

```

A B C D E F G
  A B C D E
    A B C
      A
  
```

16. 编写程序，使用双重循环输出以下图形。

```

      * * * * *
    *   * * * *   *
  * *   * * *   * *
* * *   *   * * * *
* * * *   * * * * *
* * *   *   * * * *
* *   * * * *   * *
*   * * * * *   *
    * * * * *
  
```

17. 一个球从 100 米高处自由落下，每次落地后又跳回到原高度的一半，再落下。编写程序，计算并输出它在第 10 次落地时，共经过多少米？第 10 次反弹多高？

18. 编写程序，实现从键盘输入 $n(1 \leq n \leq 9)$ ，然后输出以下内容。

```

1=1
2+22=24
3+33+333=369
...
n+nn+...+n...n=...
  
```

19. 我国有四大淡水湖。甲说：洞庭湖最大，洪泽湖最小，鄱阳湖第三。乙说：洪泽湖最大，洞庭湖最小，鄱阳湖第二，太湖第三。丙说：洪泽湖最小，洞庭湖第三。丁说：鄱阳湖最大，太湖最小，洪泽湖第二，洞庭湖第三。四个人每人仅答对了一个，请编写程序给出四个湖从大到小的顺序(提示：甲的三个判断可表示为三个逻辑表达式，而这三个逻辑表达式的值加起来应为 1，因为 C 语言中真为 1，假为 0，甲仅答对了一个)。

第6章 数 组

在前面的学习中使用的数据都属于基本数据类型，每个基本数据类型的变量只能存储一个值。在教师工资管理系统中，如果有 100 位教师，1 位教师的工资数据用 1 个变量，则存储教师的工资数据需要 100 个变量，是一件烦琐的事情。C 语言提供了数组类型，利用它可以轻松解决这个问题。数组并非一组数，而是一组变量。定义一个数组，实际上是定义了一组变量，并且这组变量的类型相同。本章将围绕一维、二维数组的定义、初始化和引用，数组的基本算法，字符数组的应用等内容进行讲解。

学习目标

- 了解什么是数组
- 掌握一维数组的定义、初始化和引用
- 掌握二维数组的定义、初始化和引用
- 掌握数组的基本算法
- 理解字符数组与字符串的概念
- 掌握字符串函数的用法，能灵活运用相应函数操作字符串

6.1 一维数组

数组是由若干相同类型数据组成的有序集合，其中每个数据又称数组元素。在程序中可利用数组来处理具有相同性质的大批数据。例如，计算平均成绩、统计总分、查找或排序等。

在数组中，每个数组元素都具有相同的数据类型，它们之间有先后顺序，可通过唯一的下标来确定数组元素的位置。通过一个下标就可以将数组中的各个元素区分开来的数组称为一维数组。

6.1.1 一维数组的定义和初始化

1. 一维数组的定义

在 C 语言中，一维数组的定义形式如下：

```
类型说明符 数组名 [常量表达式];
```

其中，“类型说明符”可是任意一种基本数据类型或构造数据类型。“数组名”是用户定义的标识符，该标识符遵循用户自定义标识符的命名规则。方括号中的“常量表达式”表示数据元素的个数，也称数组长度。数组的定义与变量的定义一样，都是为定义的对象分配存储空间。例如，定义一个有 6 个元素的整型数组变量 a 的语句如下：

```
int a[6];
```

一个好的编程习惯是在定义数组前先定义一个符号常量来表示数组元素的个数。按照这种习惯，数组 `a` 可以改写为：

```
#define NUM 6                /*定义一个符号常量表示数组元素的个数*/
int a[NUM];                 /*定义一个整型数组，其大小为 NUM*/
```

数组 `a` 的 6 个数组元素为：`a[0]`、`a[1]`、`a[2]`、`a[3]`、`a[4]`、`a[5]`。

字符数组的定义如下：

```
#define SIZE 5               /*定义一个符号常量表示数组元素的个数*/
char array[SIZE];          /*定义一个字符数组，其大小为 SIZE*/
```

数组 `array` 的 5 个数组元素为：`array[0]`、`array[1]`、`array[2]`、`array[3]`、`array[4]`。

浮点型数组的定义如下：

```
#define LENGTH 100          /*定义一个符号常量表示数组元素的个数*/
double weight[LENGTH];     /*定义一个双精度型数组，其大小为 LENGTH*/
```

图 6-1 为数组 `a` 的逻辑存储方式。对于数组 `a` 中的每个元素按照顺序依次存储，即在 `NUM` 个存储单元中，第 1 个存储单元存储第 1 个数组元素，第 2 个存储单元存储第 2 个数组元素，以此类推，直至第 `NUM` 个存储单元存储第 `NUM` 个数组元素。顺序存储是数组的一个基本特性，这个特性提供了一个简单的存储机制来存储具有线形结构特征的数据。

由于数组中的数组元素是按顺序存储的，因此任意一个数据元素可以根据数组名和该数组元素的位置来获得。这个位置称为数组元素的索引值或下标值。第 1 个数组元素的索引值是 0，第 2 个数组元素的索引值是 1，例如：

`a[0]` 表示数组中的第 1 个数组元素；

`a[1]` 表示数组中的第 2 个数组元素；

...

`a[5]` 表示数组中的第 6 个数组元素。

图 6-2 指出了数组中的每个数组元素在内存中的存储位置。数组名和下标(索引值)指出了每个数组元素在数组中的位置。虽然编译器为数组中的第 1 个元素指定的索引值看起来有些奇怪，但是这样做可以增加获取数组元素的速度。从内部机制来说，计算机将索引值作为数组元素相对于数组起始位置的偏移量。图 6-3 为获取 `a[4]` 的示意图，说明了索引值告诉计算机从数组的起始位置越过多少个元素到达目标。

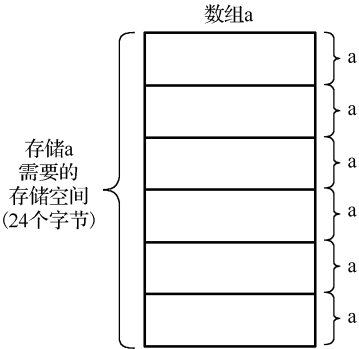


图 6-1 数组 `a` 的逻辑存储方式

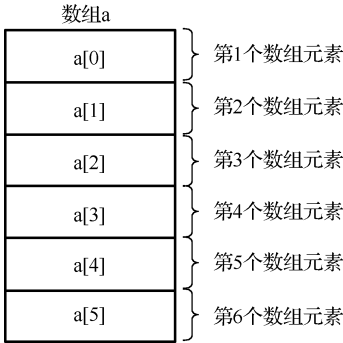


图 6-2 数组 `a` 的物理存储方式

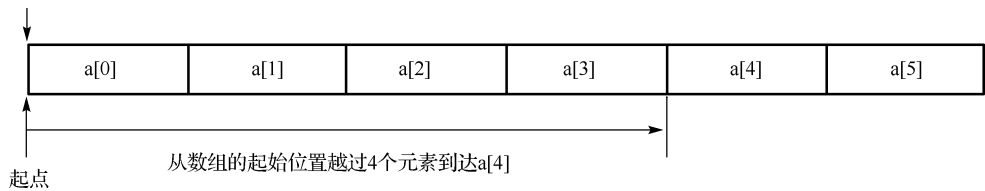


图 6-3 获取 a[4] 的示意图

综上所述，关于数组的定义要注意以下几点。

- ① 数组名的命名规则需遵循用户自定义标识符的命名规则。
- ② 说明数组大小的常量表达式必须为整型，并且只能用方括号括起来。
- ③ 说明数组大小的常量表达式不能是变量，以下程序在编译过程中会出现未知数组大小的错误。

```
#include <stdio.h>
int main()
{
    int num = 6;
    int array[num];
    return 0;
}
```

- ④ 数组名不能与其他变量名相同，以下程序在编译时会出现错误。

```
#include <stdio.h>
#define NUM 6 /*定义一个符号常量表示数组元素的个数*/
int main()
{
    int sum;
    int sum[NUM];
    return 0;
}
```

- ⑤ 数组元素的下标是从 0 开始的，以下程序说明了一个长度为 3 的一维整型数组，在这个数组中的 3 个元素分别为 array[0]、array[1]、array[2]，而没有元素 array[3]。

```
int array[3];
```

- ⑥ 允许在同一个类型定义中，定义多个数组和多个变量，例如：

```
int a,b,c[10],d[20];
```

2. 一维数组的初始化

正如变量可以在定义时被初始化一样，数组也可以，区别在于：变量的初始化仅仅需要一个值，而数组的初始化需要一系列值。一维数组的初始化可分为以下几种情况。

(1) 一般初始化操作

初始化时用一对花括号括起来，值与值之间用逗号分隔，例如：

```
int array[5] = {0,1,2,3,4};
```

初始化列表给出的值依次赋给数组的各个元素, `array[0]`被赋值为 0, `array[1]`被赋值为 1, ..., `array[4]`被赋值为 4。

如果初始值的个数大于数组定义中数组的长度, 则语法错误。例如:

```
int array[5] = {0,1,2,3,4,5};
```

对数组元素完成初始化操作后, 在程序中还可以用其他方式(如赋值语句等)重新赋值。例如:

```
array[0] = 25;
array[3] = sizeof(double);
array[4] = i++;
```

(2) 部分元素初始化

在对数组进行初始化操作时, 也可以仅对部分数组元素进行初始化操作, 例如:

```
int grades[6] = {98,87,100};
```

该语句相当于仅仅对数组 `grades` 的前 3 个数组元素进行了赋值操作, 即 `grades[0]=98`、`grades[1]=87`、`grades[2]=100`, 而 `grades[3]`、`grades[4]`、`grades[5]` 将被自动赋值为 0。因为编译器只知道初始值不够, 但它无法知道缺少的是哪些值, 所以只允许省略最后几个初始值。

当被定义的数组长度与提供初始值的个数不相同, 数组长度不能省略。若打算定义的数组长度为 10, 但是仅仅提供了 5 个初始值, 就不能省略数组长度, 而必须写成:

```
int array[10] = {0,1,2,3,4};
```

数组 `array` 的前 5 个数组元素 `array[0]`、...、`array[4]` 分别被初始化为 0、...、4, 而数组 `array` 的后 5 个数组元素 `array[5]`、...、`array[9]` 分别被初始化为 0。

如果要将数组的元素全部初始化为 0, 只需要将第 1 个元素初始化为 0, 例如:

```
int array[10] = {0};
```

(3) 省略数组长度

当对全部数组元素进行初始化操作时, 可以不指定数组长度, 例如:

```
int array[5] = {0,1,2,3,4};
```

可以写成:

```
int array[ ] = {0,1,2,3,4};
```

这是因为虽然数组的定义中并没有给出数组的长度, 但是编译器具有把所容纳的所有初始值的个数设置为数组长度的能力。

省略数组大小只能在有初始化的数组定义中, 下面的代码将产生一个编译错误:

```
int a[ ]; /*error, 没有确定数组大小*/
```

在定义数组时, 无论如何, 编译器必须知道数组的大小。

(4) 静态存储数组的自动初始化操作

一维静态存储数组的定义如下:

```
static 类型说明符 数组名 [常量表达式];
```

例如:

```
static int array[5];
```

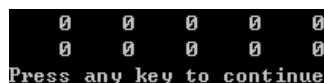
一维静态存储数组只在程序开始执行之前初始化一次。程序并不需要执行指令把这些值放到合适的位置,因为它们一开始就在那里了,这是由链接器完成的。链接器用包含可执行程序的文件中合适的值对数组元素进行初始化。如果数组未被初始化,数组元素的初始值将会自动设置为零。当这个文件载入到内存中准备执行时,初始化后的数组值和程序指令一样也被载入内存。因此,当程序执行时,静态数组已经初始化。

但是,对于自动变量而言,在默认情况下是未初始化的。如果在自动变量的定义中给出初始值,每次当执行流进入自动变量定义所在的作用域时,变量就被一条隐式的赋值语句初始化。

【案例 6-1】 定义一个具有 10 个整型数据的一维静态存储数组,每行输出 5 个数组元素。

```
#include <stdio.h>
#define N 10
int main()
{
    int i;
    static int array[N];    /*定义一个具有 N 个数组元素的静态存储数组 array*/
    for(i=0;i<N;i++)
    {
        if((i+1)%6==0)      /*每行输出 5 个数组元素*/
            printf("\n");
        printf("%5d",array[i]);    /*输出数组元素*/
    }
    printf("\n");
    return 0;
}
```

程序的运行结果如图 6-4 所示。



```
0 0 0 0 0
0 0 0 0 0
Press any key to continue
```

图 6-4 运行结果

6.1.2 一维数组的引用

数组必须先定义再使用。在 C 语言中,数组元素只能逐个引用,不能一次引用数组中的全部元素。

数组元素的引用形式如下:

```
数组名[下标]
```

【案例 6-2】 定义一个具有 5 个数组元素的整型数组 array,数组元素的值与其下标相同,并将这些数组元素输出。

```
#include <stdio.h>
#define NUM 5
int main()
{
    /*定义一个符号常量用以表示数组元素的个数*/
    static int array[NUM];
    for(i=0;i<NUM;i++)
        array[i]=i;
    for(i=0;i<NUM;i++)
        printf("%5d",array[i]);
    printf("\n");
    return 0;
}
```



```

{
    int i;
    int array[NUM];           /*定义一个整型数组，其大小为 NUM*/
    for(i=0;i<NUM;i++)
    {
        array[i]=i;           /*数组元素的赋值*/
        printf("array[%d]=%d\n",i,array[i]);    /*输出数组元素*/
    }
    return 0;
}

```

程序的运行结果如图 6-5 所示。

C 语言并不检查数组下标是否越界，这样可以提高程序的运行效率，为指针操作带来更多方便。如果定义一个数组 $a[N]$ (N 为一个符号常量)，其下标有效范围为 $[0, (N-1)]$ 。若要引用下标 N ，编译器并不提示错误，但也存在一些隐患。

【案例 6-3】 分析以下程序的运行结果。

```

array[0]= 0
array[1]= 1
array[2]= 2
array[3]= 3
array[4]= 4
Press any key to continue_

```

图 6-5 运行结果

```

#include <stdio.h>
int main()
{
    int i;
    int a[3]={1,2,3};
    for (i=1;i<=3;i++)
    {
        a[i]=0;
        printf("a[%d]=%d\n",i,a[i]);
    }
    return 0;
}

```

分析如下：

假设计算机为变量 i 分配的内存地址为 $0x0013ff7c$ (此地址刚好与 $a+3$ 相同)，数组 a 中各元素分配的内存位置如下。

```

a[0]地址: 0x0013ff70
a[1]地址: 0x0013ff74
a[2]地址: 0x0013ff78

```

当 $i=1$ 时， $a[1]$ 的值为 0； i 进行自增运算后的值为 2， $a[2]$ 的值为 0； i 再次进行自增运算后的值为 3，此时，程序将找到数组元素 $a[3]$ 所在的内存位置 (即本案例中分配给变量 i 的内存单元 $0x0013ff7c$)，并写入 0，从而导致变量 i 的值为 0。接着到 $for()$ 循环中去判断条件 $i \leq 3$ ，因为 i 的值又被置为 0， $i \leq 3$ 成立，导致再次开始执行循环，这样程序将陷入死循环。这就是 C 语言中数组不检查下标造成的隐患。

数组元素的使用方法与同类型变量的使用方法完全相同。在可以使用某种类型变量的地方都可以使用该种类型的数组元素。

定义数组时，方括号中的常量表达式表示数组的长度，这个表达式可以是整型常量，也可以是符号常量，但一定不能是变量；这也就是说在定义数组时，数组的长度必须明确指定，并且在程序的运行过程中不能更改。在数组元素引用时，方括号内的整型表达式表示数组的下标，可以是变量，下标的有效范围为 $[0, (\text{数组长度}-1)]$ 。

6.1.3 一维数组程序举例

【案例 6-4】编写程序，实现求一个具有 10 个数组元素的一维整型数组 `array[10]={65,24,58,178,190,237,-121,0,-165,6}` 中的最大值，以及最大值所在的位置。

案例分析：要求数组中的最大值，可将数组中的某个数组元素值与其他元素进行比较。使用程序实现需要定义两个变量，一个用来存储已比较的数组元素的最大值，另一个用来存储最大值所在的位置。为了实现方便且便于理解，通常将存储最大值的变量的初始值设置为数组第 1 个元素的值，相应地，将存储最大值所在位置的变量的初始值设置为 0。使用循环结构访问数组中的每个数组元素，将访问到的数组元素的值与现有的最大值进行比较，若该数组元素的值比现有的最大值大，则修改最大值，同时更改当前最大值所在的位置，否则进行下一个数组元素的比较。程序代码如下。

144

```
#include <stdio.h>
#define N 10
int main()
{
    int array[10]={65,24,58,178,190,237,-121,0,-165,6};
    int max=array[0];           /*将最大值初始化为数组的第 1 个元素*/
    int location=0,i;
    for(i=1;i<N;i++)
        if(max<array[i])       /*用当前访问到的数组元素和 max 比较，
                                如果当前访问到的数组元素大于 max，
                                则修改 max 和 location 的值*/
        {
            max=array[i];
            location=i;
        }
    for(i=0;i<N;i++)
        printf("%5d",array[i]);
    printf("\n");
    printf("最大值为:%d,是数组中的第%d个元素。\\n",max,location+1);
    return 0;
}
```

程序的运行结果如图 6-6 所示。

```
65  24  58 178 190 237 -121  0 -165  6
最大值为:237,是数组中的第6个元素。
Press any key to continue_
```

图 6-6 运行结果

【案例 6-5】编写程序，实现在教师工资管理系统中输入教师号，且不允许重复。

案例分析：因为每个教师号的数据类型都相同，所以可将教师号存入一个数组中。本案

例题目就变成了在一个数组中输入数据，不允许重复。将当前要输入的数据与在数组中已经存在的元素进行比较，如果相同，则提示重新输入，将重新输入的元素再与数组中已经存在的数组比较，如果相同，则再次输入；如果不同，则允许本次输入。重复此过程就可实现在数组中输入不同元素。程序代码如下。

```
#include <stdio.h>
#define N 10
int main()
{
    int array[10],i,j,t;
    for(i=0;i<10;i++)
    {
        printf("请输入第%d个数组元素: ",i+1);
        scanf("%d",&t);
        for(j=0;j<i;j++)
        {
            if(array[j]==t)
            {
                printf("元素已经存在,请重新输入: ");
                scanf("%d",&t);
                j=-1;
            }

        }
        array[i]=t;
    }
}
```

145

程序的运行结果如图 6-7 所示。

```
请输入第1个数组元素: 1
请输入第2个数组元素: 2
请输入第3个数组元素: 3
请输入第4个数组元素: 4
请输入第5个数组元素: 5
请输入第6个数组元素: 6
请输入第7个数组元素: 4
元素已经存在, 请重新输入: 6
元素已经存在, 请重新输入: 7
请输入第8个数组元素: 1
元素已经存在, 请重新输入: 9
请输入第9个数组元素: 10
请输入第10个数组元素: 11
Press any key to continue.
```

图 6-7 运行结果

【案例 6-6】 编写程序，实现使用冒泡排序法对具有 10 个数组元素的一维整型数组 `array[10]={65,24,58,178,190,237,-121,0,-165,6}` 中的元素按照由小到大的顺序排序，输出排序前、后的数组。

案例分析：冒泡排序法的基本思想是将相邻两个数组元素进行比较，如果前者大于后者

则将两个数组元素交换位置，即数值小的数组元素在前，数值大的数组元素在后。冒泡排序法的第 1 趟比较交换过程如图 6-8 所示。

65 24 58 178 190 237 -121 0 -165 6	原始的 10 个数
65 24 58 178 190 237 -121 0 -165 6 需要交换	第 1 次比较需要交换位置
24 65 58 178 190 237 -121 0 -165 6 需要交换	第 2 次比较需要交换位置
24 58 65 178 190 237 -121 0 -165 6 不用交换	第 3 次比较不用交换位置
24 58 65 178 190 237 -121 0 -165 6 不用交换	第 4 次比较不用交换位置
24 58 65 178 190 237 -121 0 -165 6 不用交换	第 5 次比较不用交换位置
24 58 65 178 190 237 -121 0 -165 6 需要交换	第 6 次比较需要交换位置
24 58 65 178 190 -121 237 0 -165 106 需要交换	第 7 次比较需要交换位置
24 58 65 178 190 -121 0 237 -165 6 需要交换	第 8 次比较需要交换位置
24 58 65 178 190 -121 0 -165 237 6 需要交换	第 9 次比较需要交换位置
24 58 65 178 190 -121 0 -165 6 237	第 1 趟比较交换后的数据

图 6-8 冒泡排序法第 1 趟比较交换过程

由图 6-8 可知，通过冒泡排序法的第 1 趟比较，最大数 237“下沉”到最后。冒泡排序法第 1 趟比较后的结果是：24,58,65,178,190,-121,0,-165,6,237。若要第 2 大的数“下沉”到倒数第 2 个位置，可以再使用一次冒泡排序法，称为冒泡排序法的第 2 趟比较交换，执行结果为：24,58,65,178,-121,0,-165,6,190,237。按照这样的方法，对这 10 个数据进行 9 趟比较，就可以使该数组按照由小到大的顺序排序，且最小数-165“浮”在第 1 个位置。程序代码如下。

```
#include <stdio.h>
#define N 10
int main()
{
    int array[10]={65,24,58,178,190,237,-121,0,-165,6};
    int temp,i,j; /*定义临时存储空间，供交换位置使用*/
    printf("排序前数组元素为:\n");
    for(i=0;i<N;i++)
        printf("%5d",array[i]);
    for(i=0;i<N-1;i++) /*比较趟数*/
        for(j=0;j<N-i-1;j++) /*每趟比较的次数，注意不要使数组元素越界*/
            if(array[j]>array[j+1])
            {
                temp=array[j];
                array[j]=array[j+1];
                array[j+1]=temp;
            }
}
```

```

printf("\n 排序后数组元素为:\n");
for(i=0;i<N;i++)
    printf("%5d",array[i]);
printf("\n");
return 0;
}

```

程序的运行结果如图 6-9 所示。

```

排序前数组元素为:
 65  24  58 178 190 237 -121  0 -165  6
排序后数组元素为:
-165 -121  0  6  24  58  65 178 190 237
Press any key to continue_

```

图 6-9 运行结果

【案例 6-7】 一维整型数组 array 的长度为 10，现有前 9 个数据，按从小到大的顺序排列，依次为：-65,0,21,58,78,90,98,106,124。编写程序，实现将 88 插入该数组，且插入后的数据依然按从小到大的顺序排列。

案例分析：要在有序的数组中插入某数据，使插入后的数组依然有序，就要确定插入位置。插入位置确定后，将从该位置起的数据依次向后移动，在确定插入的位置处放入该数据即可，注意数组不能越界。程序代码如下。

```

#include <stdio.h>
#define N 10
int main()
{
    int x=88,i,location;
    int array[N]={-65,0,21,58,78,90,98,106,124};
    printf("插入前: \n");
    for(i=0;i<N-1;i++)
        printf("%5d",array[i]);
    printf("\n");
    if(x>=array[8])
    {
        array[9]=x;
    }
    else
    {
        for(i=0;i<9;i++)
        {
            if(array[i]>=x)
            {
                location=i;          /*查找到要插入的位置*/
                break;
            }
        }
        for(i=N-1;i>location;i--)
            array[i]=array[i-1];      /*将要插入位置后的元素依次向后移动*/
    }
}

```

```

        array[location]=x;           /*将 x 的值赋给数组元素*/
    }
    printf("插入后: \n");
    for(i=0;i<N;i++)
        printf("%5d",array[i]);
    printf("\n");
    return 0;
}

```

程序的运行结果如图 6-10 所示。

```

插入前:
-65   0   21   58   78   90   98  106  124
插入后:
-65   0   21   58   78   88   90   98  106  124
Press any key to continue

```

图 6-10 运行结果

【案例6-8】编写程序,实现在一维整型数组 `array[10]={-165,-121,0,6,24,58,65,178,190,237}` 中查找是否存在 125, 如果存在, 输出该数在数组中的位置; 如果不存在, 输出“不存在”。

案例分析: 如果要查找某数是否存于数组中, 可使用循环结构依次取数组中的元素与给定值进行比较, 若存在, 则终止程序的执行, 返回该数值在数组中的位置; 如果不存在, 则进行下一个数组元素的比较。

```

#include <stdio.h>
#define N 10
int main()
{
    int array[N]={-165,-121,0,6,24,58,65,178,190,237};
    int location=0,i;
    for(i=0;i<N;i++)
        if(array[i]==125)
        {
            location=i;
            break;
        }
    if(i==N)           /*如果没有找到给定的值, 则退出循环, 此时 i 的值为 10*/
        printf("不存在! \n");
    else
        printf("125 在数组中的位置是: %d\n",location+1);
    return 0;
}

```

```

不存在!
Press any key to continue

```

图 6-11 运行结果

程序的运行结果如图 6-11 所示。

【案例 6-9】 如果一维数组是有序的(数组按照由大到小或者由小到大的顺序排列), 使用折半查找法(又称二分查找法)可以大大提高数据的查找效率, 折半查找法的基本思想是: 设有 N 个有序数据(以从小到大为例)存放在数组 `array[0]~array[N-1]` 中, 要查找的数据为 x , 用变量 `bottom`、`top` 和 `mid` 分别表示查

找数组下界、上界和中间位置， $\text{mid}=(\text{bottom}+\text{top})/2$ ，折半查找的算法如下：

- ① $x==a[\text{mid}]$ ，则已找到，退出循环，否则进行下面的判断；
- ② $x<a[\text{mid}]$ ， x 必定在 $\text{bottom}\sim\text{mid}-1$ 的范围之内，即 $\text{top}=\text{mid}-1$ ；
- ③ $x>a[\text{mid}]$ ， x 必定在 $\text{mid}+1\sim\text{top}$ 的范围之内，即 $\text{bottom}=\text{mid}+1$ ；
- ④ 在确定了新的查找范围后，重复进行以上比较，直到找到数据或者 $\text{bottom}>\text{top}$ 为止。

可见折半查找法每进行一次，查找范围就缩小一半。使用折半查找法的基本思想编写程序，实现在一维整型数组 $\text{array}[10]=\{-165,-121,0,6,24,58,65,178,190,237\}$ 中查找是否存在-121，如果存在，输出该数在数组中的位置；如果不存在，输出“不存在”。

```
#include <stdio.h>
#define N 10
int main()
{
    int array[N]={-165,-121,0,6,24,58,65,178,190,237};
    int x=-121;          /*待查找的数据*/
    int location;         /*记录待查找数据在数组中的位置*/
    int bottom=0;         /*下界*/
    int top=N-1;          /*上界*/
    int mid;              /*中间位置*/
    int flag=0;           /*用于标识待查找的数据是否找到，若找到，则 flag 置 1*/
    while(bottom<=top)
    {
        mid=(bottom+top)/2;
        if(array[mid]==x)
        {
            location=mid;
            flag=1;
            break;
        }
        else if(array[mid]>x)
            top=mid-1;
        else
            bottom=mid + 1;
    }
    if(flag==0)
        printf("不存在! \n");
    else
        printf("%d 在数组中的位置是: %d\n ",x,location+1);
    return 0;
}
```

程序的运行结果如图 6-12 所示。

```
-121 在数组中的位置是: 2
Press any key to continue.
```

图 6-12 运行结果

6.2 二维数组

如果某个数组的维数不止一个，这个数组就称多维数组。在多维数组中，二维数组是最常用的，本节将介绍二维数组的使用方法。

6.2.1 二维数组的定义和初始化

1. 二维数组的定义

二维数组的定义如下：

```
类型说明符 数组名 [常量表达式 1] [常量表达式 2];
```

与一维数组的定义方法一样，“类型说明符”是任意一种基本数据类型或构造数据类型，“数组名”是用户定义的标识符，该标识符遵循用户自定义标识符的命名规则。“常量表达式 1”设置二维数组的行数，“常量表达式 2”设置二维数组的列数，它们为整型常量或者计算结果为整型数值的表达式。

一个 2 行 3 列的整型数组 `array` 定义如下：

```
#define R 2
#define C 3
int array[R][C];
```

数组 `array` 具有的数据元素为：`array[0][0]`、`array[0][1]`、`array[0][2]`、`array[1][0]`、`array[1][1]`、`array[1][2]`。

由此可见，与一维数组一样，二维数组元素中的各维下标也都是从 0 开始的。

二维数组在内存中的存储形式有两种：以行序为主序和以列序为主序。以行序为主序的存储方式是按行存储，即按照第 1 行、第 2 行、…、第 $(R-1)$ 行的顺序依次存储。以数组 `array` 为例，就是先存储第 1 行的 3 个元素 `array[0][0]`、`array[0][1]`、`array[0][2]`，然后再存储第 2 行的 3 个元素 `array[1][0]`、`array[1][1]`、`array[1][2]`，如图 6-13 所示。以列序为主序的存储方式是按列存储，即按照第 1 列、第 2 列、…、第 $(C-1)$ 列的顺序依次存储。以数组 `array` 为例，就是先存储第 1 列的 2 个数组元素 `array[0][0]`、`array[1][0]`，然后存储第 2 列的 2 个数组元素 `array[0][1]`、`array[1][1]`，最后存储第 3 列的 2 个数组元素 `array[0][2]`、`array[1][2]`，如图 6-14 所示。在 C 语言中，二维数组在内存中的存储方式是以行序为主序的。

2. 二维数组的初始化

二维数组在定义时可以在“类型说明符”前使用关键字 `static` 修饰，使该数组成为静态存储的数组，此时数组中的每个元素的初始值均为 0。如果在定义二维数组时没有全部初始化数组中的元素，则没有被初始化的数组元素被赋值为 0。二维数组的初始化通常使用以下两种方法。

(1) 使用初始化列表

编写初始化列表有两种方法。

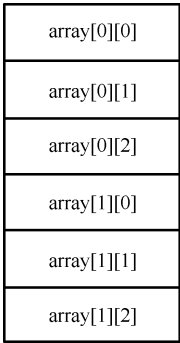


图 6-13 array 以行序为主序的存储方式

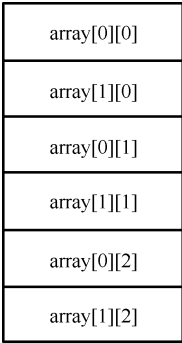


图 6-14 array 以列序为主序的存储方式

第 1 种方法是给出一个长的初始值列表，例如：

```
int matrix[2][3]={1,2,3,4,5,6};
```

二维数组的存储顺序是根据最右侧的下标率先变化的原则确定的，所以这条初始化语句等价于下列赋值语句：

```
matrix[0][0]=1; matrix[0][1]=2; matrix[0][2]=3;
matrix[1][0]=4; matrix[1][1]=5; matrix[1][2]=6;
```

第 2 种方法是基于二维数组实际上是复杂元素的一维数组这个概念，例如：

```
int two_dim[4][3];
```

可以把 `two_dim` 看成包含 4 个元素的一维数组。为了初始化这个包含 4 个元素的一维数组，使用一个包含 4 个初始值的初始化列表：

```
int two_dim[4][3]={■, ■, ■, ■};
```

但是，该数组的每个元素实际上都是包含 3 个元素的整型数组，所以每个“■”的初始化列表都应该是一个由花括号包括的 3 个整型值，将“■”用这类列表替换，可得到类似的如下代码：

```
int two_dim[4][3]={ {0,1,2},
                    {3,4,5},
                    {6,7,8},
                    {9,10,11}
                    };
```

如果没有花括号，只能在初始化列表中省略最后几个初始值，因为中间元素的初始值不能省略。使用这种方法可以为二维数组中的部分数组元素赋值，每个子初始列表都可以省略尾部的几个初始值，同时每一维初始列表各自都是一个初始化列表。例如：

```
int two_dim[4][3]={ {0,1},
                    {3},
                    {0},
                    {9,10,11}
                    };
```

等价于下列赋值语句：

```
two_dim[0][0] = 0;   two_dim[0][1] = 1;   two_dim[0][2] = 0;
two_dim[1][0] = 3;   two_dim[1][1] = 0;   two_dim[1][2] = 0;
two_dim[2][0] = 0;   two_dim[2][1] = 0;   two_dim[2][2] = 0;
two_dim[3][0] = 9;   two_dim[3][1] = 10;  two_dim[3][2] = 11;
```

说明：第 3 行的元素值都是 0，可以写成{0}或{0,0,0}，但不能写成{}，也就是说，初值的“{}”中不能为空。

(2) 自动计算数组长度

在二维数组中，只有第一维可根据初始化列表省略，而第二维必须写出，这样编译器就能推断出第一维的长度。例如：

```
int two_dim[][3]={ {0,1},
                   {3},
                   {0},
                   {9,10,11}
                 };
```

152

编译器只要统计初始化列表中包含的初始值的个数，就能推断出第一维的长度为 4。

因此，在初始化二维数组时：

① 当为全部数组元素赋初值时，说明语句中可以省略第一维的长度说明(但方括号不能省略)。例如，下列两个语句是等价的：

```
int array[2][3]={1,2,3,4,5,6};
int array[ ][3]={1,2,3,4,5,6};
```

② 在分行赋初值时也可以省略第一维的长度说明。例如，下列两个语句是等价的：

```
int array[3][3]={ {1,2},{0},{7}};
int array[ ][3]={ {1,2},{0},{7}};
```

除了使用上述两种方法，还可以使用输入函数、随机函数等为每个数组元素赋值。

【案例 6-10】 现有 3 行 5 列的二维整型数组 **matrix**，每个数组元素的值是其行坐标与列坐标的平方和，编写程序，将该二维数组输出，要求输出的也是 3 行 5 列。

案例分析：由于该案例的每个数组元素有一定的规律，因此可以使用循环结构实现，输出时要在每行输出结束处换行。程序代码如下。

```
#include <stdio.h>
#define R 3
#define C 5
int main()
{
    int matrix[R][C];           /*定义一个二维数组*/
    int i,j;
    for(i=0;i<R;i++)           /*控制行*/
        for(j=0;j<C;j++)       /*控制列*/
            matrix[i][j]=i*i+j*j;
    /*以下双重循环用于输出该二维数组*/
```

```

    for(i=0;i<R;i++)
    {
        for(j=0;j<C;j++)
            printf("%5d",matrix[i][j]);
        printf("\n");
    }
    printf("\n");
    return 0;
}

```

程序的运行结果如图 6-15 所示。

```

0   1   4   9  16
1   2   5  10  17
4   5   8  13  20
Press any key to continue

```

图 6-15 运行结果

6.2.2 二维数组的引用

与一维数组一样，二维数组也必须先定义再使用。只能逐个引用二维数组中的元素，不能一次引用二维数组中的全部元素。二维数组元素的引用形式如下：

数组名[行下标][列下标]

说明：

① 下标可以是整型常量或表达式。例如：

```
array[1][2],array[2-1][1*1]
```

② 数组元素可以出现在表达式中，也可以被赋值。例如：

```
array[1][1]=100;
array[1][2]==array[0][0]/4;
```

③ 在引用数组元素时，注意下标值必须在定义的数组大小范围内。例如：

```
int matrix[4][5];
```

在引用时，若使用了“`matrix[4][5]=88;`”，则该引用超越了数组的定义范围，即出现了越界访问。这是因为无论是行下标还是列下标都是从 0 开始的，所以行下标的取值为 0、1、2、3，而列下标的取值为 0、1、2、3、4。因此，在“`matrix[4][5]=88;`”中无论是行下标还是列下标都超出了合理的取值范围，即出现了越界。

注意：

需要区分定义数组时用的 `int matrix[4][5]` 和引用数组元素时用的 `matrix[4][5]`。定义时的 4 和 5 是用来定义数组的行的数量和列的数量；而引用数组元素时的 4 和 5 指的是该数组元素所在的数组中的行和列的值，此时 `matrix[4][5]` 指一个数组元素。

6.2.3 二维数组程序举例

【案例 6-11】 已知一个二维整型数组 `matrix[3][4]={21,32,43,56,12,89,76,70,234,30,54,88}`，求该二维数组中的最大值，以及最大值所在的行号和列号。

案例分析：对于求二维数组的最大值问题，一般的解决方案是以该数组的第 1 个元素作为最大值变量的初始值，然后依次与每个元素进行比较，如果比最大值变量大，则更改最大

值变量，并记录所在的行号和列号，直至比较完所有的数组元素。该方法也适用于求二维数组的最小值问题。程序代码如下。

```
#include <stdio.h>
#define R 3
#define C 4
int main()
{
    int i,j,row=0,column=0,max;
    int matrix[R][C]={21,32,43,56,12,89,76,70,234,30,54,88};
    max=matrix[0][0];           /*把第1个元素的值赋给max*/
    for(i=0;i<R;i++)           /*for循环次数控制行*/
        for(j=0;j<C;j++)       /*for循环次数控制列*/
            if(matrix[i][j]>max) /*循环一次，数组元素的值与max比较*/
            {
                max=matrix[i][j]; /*比较后大的元素赋给max*/
                row=i;           /*把当时比较后大的元素的行赋给row*/
                column=j;        /*把当时比较后大的元素的列赋给column*/
            }
    printf("max=%d\nrow=%d\ncolumn=%d\n",max,row,column);
    return 0;
}
```

154

```
max=234
row=2
column=0
Press any key to continue_
```

图 6-16 运行结果

程序的运行结果如图 6-16 所示。

【案例 6-12】 编写程序，实现求两个矩阵的乘积矩阵

$$C=AB, \text{ 已知: } A=\begin{Bmatrix} 2 & 4 & 6 & 8 \\ 1 & 3 & 6 & 5 \end{Bmatrix}, B=\begin{Bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{Bmatrix}, \text{ 求矩阵 } C.$$

案例分析：线性代数中的矩阵就是 C 语言中的二维数组，因此要想实现两个矩阵的乘积就必须满足第 1 个矩阵的列数与第 2 个矩阵的行数相等，然后使用线性代数中的矩阵乘法法则进行编程实现。程序代码如下。

```
#include <stdio.h>
#define L 2
#define M 4
#define N 3
int main()
{
    int i,j,k,c[L][N];
    int a[L][M]={2,4,6,8,1,3,6,5};
    int b[M][N]={1,2,3,4,5,6,7,8,9,10,11,12};
    for(i=0;i<L;i++)           /*矩阵相乘，外循环 2 次表示行*/
        for(j=0;j<N;j++)       /*内循环 3 次表示每行几列*/
        {
            c[i][j]=0;
        }
}
```

```

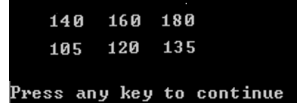
        for(k=0;k<M;k=k+1)
            c[i][j]=c[i][j]+a[i][k]*b[k][j];        /*求某一项的值*/
    }
    for(i=0;i<L;i=i+1)                                /*输出每个新的数组元素*/
    {
        for(j=0;j<N;j=j+1)
            printf("%6d",c[i][j]);
        printf("\n");
    }
    return 0;
}

```

程序的运行结果如图 6-17 所示。

【案例 6-13】 编写程序，实现将一个二维数组的行元素和列元素互换，然后存储到另一个二维数组中。

案例分析：对于二维数组行、列互换的问题，实际上是再次定义一个二维数组，新数组的行数是原数组的列数，新数组的列数是原数组的行数，然后再使用循环结构根据新、老数组的关系进行处理。程序代码如下。



```

140 160 180
105 120 135
Press any key to continue

```

图 6-17 运行结果

```

#include <stdio.h>
#define R 2
#define C 3
int main()
{
    int array[R][C]={1,2,3},{4,5,6}};
    int matrix[C][R],i,j;
    printf("array:\n");
    for(i=0;i<R;i++)
    {
        for(j=0;j<C;j++)
        {
            printf("%5d",array[i][j]);
            matrix[j][i]=array[i][j];
        }
        printf("\n");
    }
    printf("matrix:\n");
    for(i=0;i<C;i++)
    {
        for(j=0;j<R;j++)
            printf("%5d",matrix[i][j]);
        printf("\n");
    }
    return 0;
}

```

程序的运行结果如图 6-18 所示。

```
array:
  1    2    3
  4    5    6
matrix:
  1    4
  2    5
  3    6
Press any key to continue
```

图 6-18 运行结果

6.3 字符数组

用于存放字符数据的数组称为字符数组。在 C 语言中，字符数组中的一个元素只能存放一个字符。字符数组也有一维、二维、多维之分，可以参考前面章节介绍的一般数组的定义方法定义字符数组。在 C 语言中并没有提供字符串数据类型，而是以字符数组的形式来存储和处理字符串的。对于存储在字符数组中的字符串，我们可以以数组元素形式逐个处理每个字符，也可以利用字符串库函数来处理字符串。

156

6.3.1 字符数组的定义

字符数组的定义与一般数组相同，定义格式如下：

```
char 数组名[常量表达式];           /*一维字符数组*/
char 数组名[常量表达式 1][常量表达式 2]; /*二维字符数组*/
```

例如：

```
char str1[30];
```

定义了一个一维字符数组 str1，共有 30 个字符数据类型的元素，占用 30 个字节的内存。

```
char str2[5][10];
```

定义了一个二维字符数组 str2，共有 50 个字符元素，占用 50 个字节的内存。

6.3.2 字符数组的初始化

可以利用字符对字符数组初始化，在花括号中依次列出各个字符，字符要用单引号括起来，字符之间用逗号隔开。例如：

```
char c[9]={'G','o','o','d',' ','b','y','e','!'};
```

c[0]='G'、c[1]='o'、c[2]='o'、c[3]='d'、c[4]=' '、c[5]='b'、c[6]='y'、c[7]='e'、c[8]='!'。

如果花括号中提供的初值个数(即字符个数)大于数组长度，则语法错误。

在定义一维数组时，若列出了所有数组元素的初值，也可以不指定数组的大小。例如：

```
char c[ ]={'G','o','o','d',' ','b','y','e','!'};
```

字符数组 c 的大小由编译系统在编译时根据初值个数来确定，此处数组 c 的元素个数为 9。

初始化时也可以仅列出数组的前一部分元素的初始值，其余元素的初值由系统自动置 0。例如：

```
char c[14]={'G','o','o','d','','b','y','e','!'};
```

初始化后字符数组 c 的存储结构如图 6-19 所示。

G	o	o	d	空格	b	y	e	!	\0	\0	\0	\0	\0
---	---	---	---	----	---	---	---	---	----	----	----	----	----

图 6-19 字符数组 c 的存储结构

二维数组也可以按照以下形式初始化:

```
char diamond[][5]={{ ' ', ' ', ' ', '* ', ' ', ' ', ' ' }, { ' ', ' ', '* ', '* ', '* ', ' ', ' ' }, { '* ', '* ', '* ', '* ', '* ' } };
```

6.3.3 字符数组的赋值

在数组定义后对数组赋值，只能通过对其中的每个元素逐个赋值的方式进行。

例如，一维数组的赋值如下：

```
char c[9];  
c[0]='G';c[1]='o';c[2]='o';c[3]='d';c[4]=' ';c[5]='b';c[6]='y';c[7]='e';c[8]='!';
```

二维数组的赋值如下:

```
char s[2][3];
s[0][0]='h'; s[0][1]='e'; s[0][2]='\0'; s[1][0]='l'; s[1][1]='o'; s[1][2]='\0';
```

用多条赋值语句依次给一维数组、二维数组的各个元素赋值，赋值后数组的存储结构如图 6-20 所示。

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]
G	o	o	d	空格	b	y	e	!

s[0][0]	s[0][1]	s[0][2]	s[1][0]	s[1][1]	s[1][2]
h	e	l	l	o	!

图 6-20 一维字符数组 c 和二维字符数组 s 的存储结构

若定义之后在赋值语句中只给部分元素赋值，则剩余没有赋值的数组元素为随机字符。
例如：

```
char c[14];
c[0]='G';c[1]='o';c[2]='o';c[3]='d';c[4]=' ';c[5]='b';c[6]='y';c[7]='e';c[8]='!';
```

c[9]~c[13]的值为随机字符。

注意：赋值时的字符个数应小于或等于字符数组的大小，否则编译时将出现语法错误。

如果数组内的元素具有某种规律性，还可以使用循环语句为字符数组赋值，这种赋值方式比较简洁。

例如，将一个数组赋值为'a'到'z'的程序如下：

```
int i;
char str[N];
for(i=0;i<26;i++)
{
    str[i]='a'+i;
}
```

由于字符型和整型通用,也可使用整型数组来存储字符。但由于 `int` 型数据类型占用 4 个字节的存储空间,而 `char` 型占用 1 个字节的存储空间,因此,使用 `int` 型数组会浪费空间。例如:

```
int s[10];
s[0]= 'c';    /*合法,但浪费存储空间*/
```

6.4 字符串

6.4.1 字符串常量

字符串常量是由一对双引号括起来的字符序列,如"john"、"I am happy"、"-12 34"。

需要注意的是:C 语言中并没有提供字符串数据类型,而是以字符数组的形式来存储和处理字符串的。系统对字符串常量自动加一个'\0'作为结束标志。例如,"C Program"共有 9 个字符,但在内存中占 10 个字节,最后一个字符'\0'是系统自动加上的。

有了结束标志'\0'后,在程序中往往依靠检测'\0'的位置来判断字符串是否结束,而不是根据数组的长度来决定字符串长度。在实际应用中,人们关心的是有效字符串的长度而不是字符数组的长度。当然,在定义字符数组时应估计实际字符串长度,保证数组长度始终大于字符串实际长度。

6.4.2 利用字符串对字符数组初始化

对使用 C 语言处理字符串的方法了解后,字符数组初始化的方法又多了一种,即用字符串常量来初始化字符数组。

用字符串给字符数组初始化是最常用的方法之一。与用字符初始化的方法相比,其表达更加简捷,可读性强,尤其便于后续数据处理,因为系统在字符串常量后自动增加一个字符串结束标志'\0',为之后对这些字符串数据的处理设置了明确的数据处理边界。例如:

```
char s[12]={"Good bye!"};
```

也可省略花括号直接写成:

```
char s[12]="Good bye!";
```

注意:用字符串对字符数组初始化与用字符对数组初始化不同,系统会在字符串常量最后自动添加一个字符串结束标志'\0',初始化后数组的存储结构如图 6-21 所示。数组的前 9 个字符为'G'、'o'、'o'、'd'、' '、'b'、'y'、'e'、'!',第 10 个字符为'\0',后 2 个元素也设定为空字符。

G	o	o	d	空格	b	y	e	!	\0	\0	\0
---	---	---	---	----	---	---	---	---	----	----	----

图 6-21 初始化后数组的存储结构

当用字符串给字符数组初始化时，可以不指定字符数组的大小，例如：

```
char s[]="Good bye!";
```

此时数组 s 的元素个数为 10，比实际字符串中的字符个数大 1，因为字符串最后自动增加了一个结束标志('\0')。

通常将一个字符串存放在一维字符数组中，多个字符串则可以存放在二维字符数组中。此时，数组第一维的长度代表存储的字符串的个数，可以省略，但第二维的长度不能省略。例如：

```
char t[][9]={"China", "American", "Japan", "Russia"};
```

定义的二维字符数组有 4 行，其存储结构如图 6-22 所示。

C	h	i	n	a	\0	\0	\0	\0
A	m	e	r	i	c	a	n	\0
J	a	p	a	n	\0	\0	\0	\0
R	n	s	s	i	a	\0	\0	\0

图 6-22 二维字符数组 t 的存储结构

注意：上述这种字符数组的整体赋值，只能在字符数组的初始化时使用，不能用于字符数组的赋值，字符数组的赋值只能对其元素一一赋值。

用以下方法对字符数组赋值是错误的：

```
char str[14];
str="I love China";
```

6.4.3 字符数组与字符串的输入、输出

对于一般数组元素的引用，只能逐个引用数组元素，而不能一次引用整个数组。对于字符数组，可以逐个字符引用，也可以将整个字符数组一次输入和输出。

1. 用格式符%c 逐个输入和输出字符

用格式符%c 逐个输入和输出字符的案例如下。

【案例 6-14】 编写程序，实现按数组元素输入与输出字符。

```
#include <stdio.h>
int main()
{
    char c[20];
    int i;
    for (i=0;i<20;i++)
        scanf("%c",&c[i]);    /*从键盘输入每个数组元素*/
    for(i=0;i<20;i++)
        printf("%c",c[i]);
    return 0;
}
```

程序的运行结果如图 6-23 所示。

本案例运用 for 循环结合格式符%c 进行数组元素的输入和输出，与一般数据类型数组的输入和输出方式无区别。

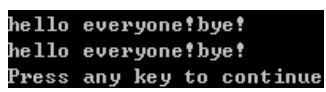


图 6-23 运行结果

2. 将整个字符串一次输入和输出

由于 C 语言中以字符数组的形式来存储和处理字符串，所以处理字符数组输入和输出的方法又增加了一种，即将整个字符串一次输入和输出。

字符串的输入与输出库函数共有 4 个，如表 6-1 所示。

表 6-1 字符串的输入与输出库函数

输 入	输 出
gets()	puts()
scanf()	printf()

在表 6-1 列出的函数中，gets() 和 puts() 用于字符串整体的输入与输出。scanf() 和 printf() 通常情况下可以代替 gets() 和 puts()，用于字符串整体的输入与输出。在程序中调用这些函数时需包含头文件 stdio.h。下面详细介绍这 4 个函数。

160

(1) gets() 函数

gets() 函数的调用格式为：

```
gets(字符数组名);
```

功能：接收键盘的输入，将输入的字符串包含空格字符存放在字符数组中，直到遇到回车符时返回。

注意：回车换行符'\n'不会作为有效字符存储到字符数组中，而是转换为字符串结束标志'\0'来存储。

【案例 6-15】 使用 gets() 函数输入一个字符串后，再将其输出到屏幕上。

```
#include <stdio.h>
int main()
{
    char line[81];
    printf("Input a string: ");
    gets(line);
    printf("The line entered was: %s\n",line);
    return 0;
}
```

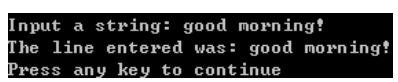


图 6-24 运行结果

程序的运行结果如图 6-24 所示。

注意：用于接收字符串的字符数组定义时的长度应足够长，以便保存整个字符串和字符串结束标志。否则，函数将把超过字符数组定义的长度之外的字符顺序保存在数组范围之外的内存单元中，从而可能覆盖其他内存变量的内容，造成程序出错。

例如：

```
char c[20];
gets(c);
```

运行程序时，只能输入不超过 19 个字符。

用 gets() 函数接收字符串时，无法限制输入字符串的长度，只能根据需要，定义一个足够大的字符数组。

(2)scanf() 函数

scanf() 函数将输入的字符保存到字符数组中，遇到空格符或回车符时终止输入操作。

在输入字符串时使用%s 格式控制符，并且与%s 对应的地址参数应该是一个字符数组名，任何时候都将忽略前导空格，scanf() 函数会自动在字符串后面加'\0'。例如：

```
char str[15];
scanf("%s",str); /*不要写成&str，因为数组名 str 是地址*/
```

当输入 “Good evening!↵” 时，str 中的字符串将是 “Good”。

注意：与 gets() 函数不同，系统会把空格符作为 scanf() 函数输入的字符串之间的分隔符，因此只将空格前的 Good 送到 str 中。由于把 Good 作为一个字符串处理，故在其后加'\0'。数组 str 的存储结构如图 6-25 所示。

G	o	o	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

图 6-25 数组 str 的存储结构

利用 scanf() 函数可以连续输入多个字符串，输入时，字符串间用空格分隔。例如：

```
char str1[10],str2[10];
scanf("%s%s",str1,str2);
```

当输入 “Good night!↵” 时，str1 中的字符串是 “Good”，str2 中的字符串是 “night!”。输入后数组 str1 与数组 str2 的存储结构如图 6-26 所示。数组中未被赋值的元素的值自动置'\0'。

G	o	o	d	\0	\0	\0	\0	\0	\0
n	i	g	h	t	!	\0	\0	\0	\0

图 6-26 数组 str1 与数组 str2 的存储结构

为了避免输入的字符串长度超过数组的大小，可以在调用 scanf() 函数时使用%ns 格式控制符，整数 n 表示域宽限制，如果没有遇到空格字符，那么读入操作将在读入 n 个输入字符之后停止。例如：

```
char str[10];
scanf("%9s",str);
```

读入字符串到字符数组 str 中，最多可读入 9 个非空格字符到 str 中，str 中的最后一个数据单元用于存放字符串结束标志'\0'。

表 6-2 给出了 gets() 函数和 scanf() 函数输入字符串的区别。

表 6-2 gets() 函数和 scanf() 函数输入字符串的区别

gets() 函数	scanf() 函数
输入的字符串中可包含空格字符	输入的字符串中不可包含空格字符
只能输入一个字符串	可连续输入多个字符串(使用%s%s)
不可限定字符串的长度	可限定字符串的长度(使用%ns)
遇到回车符时结束	遇到空格符或回车符时结束

(3) puts() 函数

puts() 函数的调用格式如下：

```
puts(字符数组名);
```

功能：将字符串中的所有字符输出到终端上，输出时将字符串结束标志'\0'转换成换行符'\n'。使用 puts() 函数输出字符串时无法进行格式控制。

【案例 6-16】 将 gets() 函数读到的字符串改用 puts() 函数输出。

```
#include <stdio.h>
int main()
{
    char line[81];
    printf("Input a string: ");
    gets(line);
    puts(line);
    return 0;
}
```

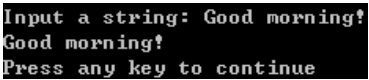


图 6-27 运行结果

程序的运行结果如图 6-27 所示。

可看到字符串输出后自动进行了换行。

(4) printf() 函数

printf() 函数在输出字符串时使用%s 格式控制符，并且与%s 对应的地址参数必须是字符串第 1 个字符的地址。printf() 函数将依次输出字符串中的每个字符，直到遇到字符'\0'，'\0'不会被输出。printf() 函数输出字符串的语句如下：

```
char s[ ]="I love china!";
printf("the string is:%s\n",s); /*等价于printf("the string is:%s\n",&s[0]);*/
printf("the second word is:%s\n",&s[2]);
printf("the last word is:%s\n","china");
```

程序的运行结果如图 6-28 所示。

printf() 输出字符串时还可以定义更多的格式。%ns 可以同时指定字符串显示的宽度。如果字符串的实际长度小于 n 个字符，不足部分填充空格。n 为正数，则在左端补空格，字符串右对齐。n 为负数，则字符串左对齐。如果字符串的实际长度大于 n 个字符，则显示整个字符串。例如：

```
printf(">>%10s<<\n","china!");
printf(">>%-10s<<\n","china!");
printf(">>%13s<<\n","I love china!");
```

程序的运行结果如图 6-29 所示。

```
the string is:I love china!  
the second word is:love china!  
the last word is:china  
Press any key to continue
```

图 6-28 运行结果

```
>> china!<<  
>>china! <<  
>>I love china!<<  
Press any key to continue
```

图 6-29 运行结果

6.4.4 字符串处理函数

C 语言的函数中提供了相当多的字符串处理函数,熟练掌握这些函数的使用,可提高编程效率。计算机处理的信息中存在非数值型的数据,例如,在学生信息的处理中,学生的姓名、性别、联系电话、爱好、家庭住址等都是用字符数据或字符串数据来表示的,那么对这些非数值型数据的处理必定要用到字符串的一些操作。

在使用字符串处理函数时,应包含头文件 `string.h`。

1. 求字符串长度的函数

调用格式如下:

```
strlen(字符串的地址);
```

功能: 返回字符串中包含的字符个数(不包含 `\0`), 即字符串的长度。

字符串的长度是指从给定的字符串的起始地址开始到第 1 个 `\0` 为止。例如:

```
char str[ ]="I love china!";  
printf("%d",strlen(str));          /*输出结果为 13*/  
printf("%d",strlen(&str[7]));      /*输出结果为 6*/
```

又如:

```
char str[ ]="I love\0china!";  
printf("%d",strlen(str));          /*输出结果为 6*/  
printf("%d",strlen(&str[7]));      /*输出结果为 6*/
```

运算符 `sizeof` 也可以计算字符串长度,但它包括该字符数组中的所有 `\0` 字符。

2. 字符串连接函数

调用格式如下:

```
strcat(字符数组 1, 字符串 2);
```

功能: 连接两个字符串,把字符串 2 连接到字符数组 1 的字符串后面,连接的结果放在字符数组 1 中,函数调用后的返回值为字符数组 1 的地址。其中字符串 2 可以是字符数组名,也可以是字符串常量。

【案例 6-17】 编写程序,实现两个字符串的连接。

```
#include<string.h>  
#include<stdio.h>  
int main()
```

```

{
    char c1[20]="I am";
    char c2[ ]=" a boy";
    printf("%s",strcat(c1,c2));
    return 0;
}

```

程序的运行结果如图 6-30 所示。




图 6-30 运行结果

注意：在定义字符数组 1 时，长度要足够大，要能容纳连接后的新字符串。连接之前，两个字符串的后面都有一个'\0'，连接时，字符串 1 后面的'\0'被字符串 2 的第 1 个字符取代，只在新串最后保留一个'\0'。

3. 字符串复制函数

调用格式如下：

```
strcpy(字符数组 1, 字符串 2);
```

功能：将字符串 2 复制到字符数组 1 中(包括字符串 2 的结束标志'\0')。字符数组 1 必须是一个字符数组变量，且其长度足够大，以便能容纳字符串 2。字符串 2 可以是字符数组名，也可以是字符串常量。例如：

```

char s1[7]="bright",s2[10]="red\0car",s3[10];
printf("%s\n",strcpy(s1,s2));          /*输出结果: red*/
strcpy(s3,"car");                      /*把"car"复制到 s3 中*/
printf("%s\n",s3);                     /*输出结果: car*/

```

说明：复制时，字符数组 2 最后的结束标志'\0'被一起复制到字符数组 1 中。不能用赋值语句将一个字符串常量或字符数组直接赋给一个字符数组。例如，以下两条语句都是不合法的：

```

str1={"good"};
str2=str1;

```

4. 字符串比较函数

调用格式如下：

```
strcmp(字符串 1, 字符串 2);
```

功能：比较字符串 1 和字符串 2。

字符串的比较规则是：对两个字符串中的字符按照自左向右的顺序逐个对照 ASCII 码大小进行比较，直到出现不同的字符或遇到'\0'为止。如果全部字符均相同，则认为两个字符串相等；若出现不同的字符，则以第 1 个不同字符的比较结果作为两个字符串的比较结果，并由函数值返回。

若字符串 1=字符串 2，函数值为 0；若字符串 1>字符串 2，函数值为一个正整数；若字符串 1<字符串 2，函数值为一个负整数。

例如：

```
char s1[]="abcde",s2[10]="abcde";
if(strcmp(s1,s2)==0)          /*比较字符串 s1、s2 结果是否等于 0*/
    printf("yes");           /*输出结果为：yes*/
```

5. 其他常用的字符串处理函数

其他一些常用的字符串处理函数如表 6-3 所示。

表 6-3 其他常用字符串处理函数

函数的用法	函数的功能	应包含的头文件
strchr(字符串,字符)	在字符串中查找第 1 次出现指定字符的位置	string.h
strstr(字符串 1,字符串 2)	查找字符串 2 在字符串 1 中第 1 次出现的位置	string.h
strlwr(字符串)	将字符串中的所有字符转换成小写字符	string.h
strupr(字符串)	将字符串中的所有字符转换成大写字符	string.h
atoi(字符串)	将字符串转换成整型	stdlib.h
atol(字符串)	将字符串转换成长整型	stdlib.h
atof(字符串)	将字符串转换成浮点型	stdlib.h

注意：库函数并非 C 语言本身的组成部分，而是人们为了使用方便而编写的公用函数，每个系统提供的函数数量，以及函数名、函数功能、参数等都可能有所不同，使用时应查阅 C 语言编译系统提供的库函数手册。

6.4.5 字符串程序举例

【案例 6-18】 编写程序，实现输入三个字符串，将其按照从小到大的顺序输出。

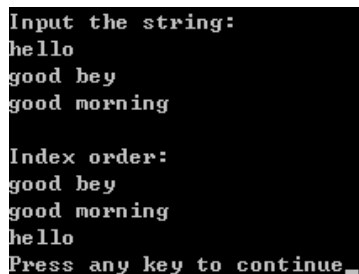
案例分析：案例中输入的三个字符串可能会含有空格，为了避免出错，最好使用 gets() 函数接收字符串，排序时字符串的比较和赋值则可以使用 strcmp() 和 strcpy() 函数。程序代码如下。

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s[50],s1[50],s2[50],s3[50];
    printf("Input the string:\n");
    gets(s1);    /*输入字符串*/
    gets(s2);    /*输入字符串*/
    gets(s3);    /*输入字符串*/
    if(strcmp(s1,s2)>0)
    {
        strcpy(s,s1);    /*复制字符串*/
        strcpy(s1,s2);    /*复制字符串*/
        strcpy(s2,s);    /*复制字符串*/
    }
    if(strcmp(s1,s3)>0)
    {
```

```

        strcpy(s,s1);
        strcpy(s1,s3);
        strcpy(s3,s);
    }
    if(strcmp(s2,s3)>0)
    {
        strcpy(s,s2);
        strcpy(s2,s3);
        strcpy(s3,s);
    }
    printf("\nIndex order:\n%s\n%s\n%s\n",s1,s2,s3);
    return 0;
}

```



```

Input the string:
hello
good bey
good morning

Index order:
good bey
good morning
hello
Press any key to continue

```

图 6-31 运行结果

程序的运行结果如图 6-31 所示。

当然，本案例也可采用二维字符数组进行处理。

【案例 6-19】 编写程序，实现输入三个字符串，输出最大字符串。

案例分析：运用二维字符数组来处理多个字符串，定义一个二维字符数组 `c[3][20]`。则 `c[0]` 是第 1 个串在内存中存放的首地址，`c[1]` 是第 2 个串在内存中存放的首地址，`c[2]` 是第 3 个串在内存中存放的首地址。`str` 是字符串在内存中存放的首地址。

利用字符串函数，先求出两个字符串中大的，并把这个大的字符串赋给串 `str`，串 `str` 再和第 3 个比较，大的字符串再赋给串 `str`，最后输出串 `str` 即可。程序代码如下。

```

#include<string.h>
#include<stdio.h>
int main()
{
    char str[20],c[3][20];
    int i;
    for (i=0;i<3;i++)
        gets(c[i]);
    if (strcmp(c[0],c[1])>0) /*比较前两个字符串的大小，把大的赋给串 str*/
        strcpy(str,c[0]);
    else
        strcpy(str,c[1]);
    if (strcmp(c[2],str)>0)
        strcpy(str,c[2]);
    printf ("max=%s\n",str);
    return 0;
}

```

程序的运行结果如图 6-32 所示。

【案例 6-20】 编写程序, 实现输入一个数字串, 将其转换为相应的整数输出。例如, 输入数字串“-1234”, 转换为整数-1234 输出; 输入数字串 “1234”, 转换为整数 1234 输出。

案例分析: 符号位 “+” 或 “-” 不能参与运算, 把它们转换为 1 或-1, 再乘以运算结果即可。程序代码如下。

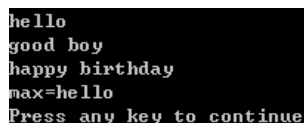


图 6-32 运行结果

```
#include<string.h>
#include<stdio.h>
int main()
{
    char s[20];
    int i,n,sign;
    i=0;
    scanf("%s",s);
    sign=(s[i]=='-')?-1:1;          /*先把正、负符号变为 1 或-1*/
    if(s[i]=='+'||s[i]=='-') i++;    /*把符号位排除到计算之外*/
    for(n=0;s[i]>='0'&&s[i]<='9';i++)
        n=10*n+s[i]-'0';
    printf("Result is %d \n",n*sign);
    return 0;
}
```

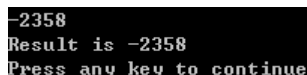


图 6-33 运行结果

程序的运行结果如图 6-33 所示。

【案例 6-21】 编写程序, 实现输入一行由字母和空格组成的字符串, 统计该串中单词的个数(假设单词之间用一个或多个空格分隔, 但第一个单词之前和最后一个单词之后可能没有空格)。

案例分析: 按照题意, 连续的一段不含空格的字符串就是单词。将一个或连续的若干个空格都作为一个空格, 那么单词的个数可以由空格出现的次数来决定。如果当前字符是非空格字符, 而它的前一个字符是空格, 则单词数量加 1; 如果当前字符是非空格字符, 而前一个字符也是非空格字符, 则单词数量不变。程序代码如下。

```
#include<stdio.h>
#define N 81
int main()
{
    char str[N];          /*在 DOS 模式下一行最多 80 个字符*/
    char ch;
    int i,num=0,word=0;    /*word 作为一个标志位, num 统计单词的个数*/
    printf("Input the string:");
    gets(str);            /*输入字符串*/
    for(i=0;(ch=str[i])!='\0';i++) /*先将 str[i] 的值赋予变量 ch, 再判断 ch
    {                          的值是否为 '\0'*/
        if(ch==' ')
            word=0;          /*当前字符为空格时, word=0*/
        else if(word==0)
            num++;
    }
```

```

        {
            word=1;                /*新单词开始*/
            num++;
        }
    }
    printf("There are %d words in the line.\n",num);
    return 0;
}

```

程序的运行结果如图 6-34 所示。

```

Input the string:hello my name is Mary I am a boy I am fourteen years old I like
swigmin and singing
There are 19 words in the line.
Press any key to continue

```

图 6-34 运行结果

说明：程序中的变量 word 作为一个标志位，当遇到一个或多个空格时，word=0；当遇到第一个非空格时，若原 word 是 0，表示新词开始，num 增 1，同时 word=1。这样循环往复操作，一直到完成计数为止。

168

6.5 本章小结

本章首先对一维、二维数组的定义、初始化和引用等相关知识进行详细讲解，并列举了相关的案例算法。灵活掌握数组的相关知识，有助于教师工资管理系统相关功能的实现。随后讲解了字符数组与字符串的定义、初始化和引用，字符串的输入和输出，以及操作字符串的相关函数等。

6.6 习题

一、单选题

- “int a[4]={5,3,8,9};”中 a[3]的值为()。
 - 5
 - 3
 - 8
 - 9
- 以下 4 个字符串函数中，()所在的头文件与其他 3 个不同。
 - gets
 - strcpy
 - strlen
 - strcmp
- 以下 4 个数组定义中，()是错误的。
 - int a[7];
 - #define N 5 long b[N];
 - char c[5];
 - int n,d[n];
- 对字符数组进行初始化，()形式是错误的。
 - char c1[]={'1','2','3'};
 - char c2[]=123;
 - char c3[]={'1','2','3','\0'};
 - char c4[]="123";

5. 在数组中, 数组名表示()。

- A. 数组第1个元素的首地址 B. 数组第2个元素的首地址
C. 数组所有元素的首地址 D. 数组最后一个元素的首地址

6. 若有以下数组说明, 则数值最小的和最大的元素的下标分别是()。

```
int a[12]={1,2,3,4,5,6,7,8,9,10,11,12};
```

- A. 1、12 B. 0、11 C. 1、11 D. 0、12

7. 若有以下说明, 则数值为4的表达式是()。

```
int a[12]={1,2,3,4,5,6,7,8,9,10,11,12}; char c='a',d,g;
```

- A. a[g-c] B. a[4] C. a['d'-'c'] D. a['d'-c]

8. 设有定义 “char s[12]="string";”, 则 “printf("%d\n",strlen(s));” 的输出是()。

- A. 6 B. 7 C. 11 D. 12

9. 设有定义 “char s[12]= "string";”, 则 printf("%d\n",sizeof(s));” 的输出是()。

- A. 6 B. 7 C. 11 D. 12

10. 以下合法的数组定义是()。

- A. char a[]="strin"; B. int a[5]={0,1,2,3,4,5};
C. char a="string"; D. char a[]={0,1,2,3,4,5}

11. 以下合法的数组定义是()。

- A. int a[3][]={0,1,2,3,4,5}; B. int a[][3]={0,1,2,3,4};
C. int a[2][3]={0,1,2,3,4,5,6}; D. int a[2][3]={0,1,2,3,4,5};

12. 下列语句中, 正确的是()。

- A. char a[3][]={'abc','1'}; B. char a[][3]='abc','1';
C. char a[3][]={'a',"1"}; D. char a[][3]={"a","1"};

13. 下列定义的字符数组中, “printf("%s\n",str[2]);” 的输出是()。

```
static str[3][20]={"basic","foxpro","windows"};
```

- A. basic B. foxpro C. windows D. 输出语句出错

14. 下列语句定义了数组, 其中()是不正确的。

- A. char a[3][10]={"China","American","Asia"};
B. int x[2][2]={1,2,3,4};
C. float x[2][]={1,2,4,6,8,10};
D. int m[][3]={1,2,3,4,5,6};

15. 数组定义为 int a[3][2]={1,2,3,4,5,6}, 则值为6的数组元素是()。

- A. a[3][2] B. a[2][1] C. a[1][2] D. a[2][3]

16. 下面程序()错误。

```
#include <stdio.h>
main()
{
    float array[5]={0.0};           //第A行
```

```

int i;
for(i=0;i<5;i++)
scanf("%f",&array[i]);
for(i=1;i<5;i++)
array[0]=array[0]+array[i];    //第B行
printf("%f\n",array[0]);      //第C行
}

```

A. 第 A 行 B. 第 B 行 C. 第 C 行 D. 没有

17. 下面()是不正确的字符串赋值或赋初值的方式。

A. char *str; str="string"; B. char str[7]={ 's','t','r','i','n','g' };
 C. char str1[10]; str1="string"; D. char str1[]="string", str2[]="12345678";

18. strlen(s) 为求字符串 s 的长度的函数, 以下语句的输出结果是()。

```

char s[12]="a book!";
printf("%d",strlen(s));

```

A. 12 B. 8 C. 7 D. 11

19. strlen(s) 为求字符串 s 的长度的函数, 以下语句的输出结果是()。

```

char sp[]="\t\v\\0will\n";
printf("%d",strlen(sp));

```

A. 14 B. 3 C. 9 D. 字符串中有非法字符

20. 以下语句的输出结果是()。

```

char str[]="\c:\\abc.dat\\";
printf("%s",str);

```

A. 字符串中有非法字符 B. \c:\\abc.dat\
 C. "c:\\abc.dat" D. "c:\\abc.dat"

21. 以下能正确进行字符串赋值操作的语句是()。

A. char str[5]={ "China" }; B. char str[5]={ 'C','h','i','n','a' };
 C. char *str; str="China"; D. char *str; scanf("%s",str);

22. 以下程序的运行结果是()。

```

char a[7]="ABCDEFGH";
char b[4]="abc";
strcpy(a,b);
printf("%c",a[5]);

```

A. 空格 B. \0 C. E D. F

23. 以下程序的运行结果是()。

```

for(i=0;i<2;i++)
for(j=0;j<2;j++)
if(i==j) {a[i][j]=1;printf("%d",a[i][j]);}
else {a[i][j]=0;printf("%d",a[i][j]);}
printf("\n");

```

A. 1 0 0 1

B. 0 1 1 0

C. 1 0

D. 0 1

0 1

1 0

二、填空题

1. C 语言中,数组的各元素必须具有相同的_____,元素的下标下限为_____,下标必须是正整数、0 或_____,但在程序执行过程中,不检查元素下标是否_____。

2. C 语言中,数组在内存中占一片_____的存储区,_____代表它的首地址。数组名是一个_____常量,不能对它进行赋值运算。

3. 执行“static int b[5],a[][3]={1,2,3,4,5,6};”后 b[4]=_____, a[1][2]=_____。

4. 设有定义语句“static int a[3][4]={ {1},{2},{3} };”,则 a[1][0]=_____, a[1][1]=_____, a[2][1]=_____。

5. 如定义语句为“char a[]="windows",b[]="7";”,则语句“printf("%s",strcat(a,b));”的输出结果为_____。

6. 根据以下说明,写出正确的说明语句。

(1)men 是一个有 10 个整型元素的数组。说明语句为_____。

(2)step 是一个有 4 个实型元素的数组,元素值分别为 1.9, -2.33, 0, 20.6。说明语句为_____。

(3)grid 是一个二维数组,共有 4 行 10 列整型元素。说明语句为_____。

7. array 是一个一维整形数组,有 10 个元素,前 6 个元素的初值是 9,4,7,49,32,-5,正确的说明语句为_____。该数组下标的取值范围是从_____到_____ (从小到大)。用 scanf() 函数输入数组的第 2 个元素,表示为_____。用赋值语句把 39 存入第 1 个元素,表示为_____。把第 6 个和第 4 个元素之和存入第 1 个元素,表示为_____。

8. 写出以下初始化数组的长度。

(1)int chn[3];

数组 chn 的长度为_____。

(2)float isa[]={1.0,2.0,3.0,4.0,5.0};

数组 isa 的长度为_____。

(3)int doom[8];

数组 doom 的长度为_____。

(4)float pci[4][2];

数组 pci 的长度为_____。

(5)int ast[3][3];

数组 ast 的长度为_____。

(6)int att[3][4];

数组 att 的长度为_____。

(7)float dell[][3]={ {1,4,7},{2,5},{3,6,9} };

数组 dell 的长度为_____。

9. 若有以下整型 a 数组,数组元素和元素的值如下所示。

数组元素: a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

元素的值: 9 4 12 8 2 10 7 5 1 3

- (1) 写出该数组的说明, 并赋以上初值。程序为_____。
 - (2) 该数组的最小下标值为_____, 最大下标值为_____。
 - (3) $a[a[9]]$ 的值为_____, $a[a[4]+a[8]]$ 的值为_____。
10. 字符串 "ab\n\\012\\\" 的长度为_____。
11. 以下程序以每行 10 个数据的形式输出 a 数组, 请填空。

```
void main()
{
    int a[50], i;
    printf("输入 50 个整数:");
    for(i=0; i<50; i++) scanf("%d", _____);
    for(i=1; i<=50; i++)
    {
        if(_____)
            printf("%3d\n", _____) ;
        printf("%3d", a[i-1]);
    }
}
```

172

12. 以下程序的功能是: 输出数组 s 中最大元素的下标, 请填空。

```
void main()
{
    int k, p;
    int s[ ]={1, -9, 7, 2, -10, 3};
    for(p=0, k=p; p<6; p++)
        if(s[p]>s[k]) _____;
    printf("%d\n", k);
}
```

13. 以下程序的功能是: 输入 20 个数存放在一个数组中, 并且输出其中最大者、最小者、20 个数的和, 以及它们的平均值, 请填空。

```
void main()
{
    char array[_____];
    int max, min, average, sum;
    int i;
    for(i=0; i<_____; i++)
    {
        printf("请输入第%d个数:", i+1);
        scanf("%d", _____);
    }
    min=max=array[0];
    for(i=0; i<=_____; i++)
    {
        if(max<array[i])
```

```

        _____;
        if(min>array[i])
            _____;
        sum=_____;
    }
    average =_____;
    printf("20 个数中最大值是%d, ",max);
    printf("最小值是%d, ",min);
    printf("和是%d, ",sum);
    printf("平均值是%d.\n",average);
}

```

14. 以下程序的功能是：将 10~99 之间的数字中个位数字与十位数字之和为奇数的所有数字存储在数组 a 中，并每行输出 5 个，再输出所有奇数的平均值，请填空。

```

#include <stdio.h>
void main(void)
{
    int a[100]={0},m,n,i,j,s=0,k=0;
    double ave;
    for(i=10,j=0;i<=99;i++)
    {
        m=_____
        n=i%10;
        if((m+n)%2)
        {
            a[k]=i;
            s+=_____;
            k++;
        }
    }
    for(i=0;i<k;i++)
    {
        printf("%4d",a[i]);
        if(_____)
            printf("\n");
    }
    ave=s*1.0/k;
    printf("\nk=%d,ave=%.2f\n",k,ave);
}

```

15. 以下程序的功能是：将字符串中下标为奇数的字符右移到下一个奇数位置，最后面被移出字符串的字符回放到第 1 个奇数的位置，下标为偶数的字符不动，如原字符串为 ABCDEFGH，处理后的字符串为 AHCBEDGF，请填空。

```

#include <stdio.h>
void main( void )
{

```

```

char ch,str[100];
int i,k,n=0;
printf("The string is:");
gets(str);
for(i=0;str[i]!='\0';i++)
    n++;
if(_____)
    k=n-1;
else
    k=n-2;
ch=_____;
for(i=k-2;i>=1;i=i-2)
    _____=str[i];
str[1]=ch;
printf("The result is:%s\n",str);
}

```

16. 以下程序的功能是: _____。

174

```

#include <stdio.h>
void main()
{
    char s[80];
    int i;
    for(i=0; i<80; i++)
    {
        s[i]=getchar();
        if(s[i]=='\n') break;
    }
    s[i]='\0'; i=0;
    while(s[i])
        putchar(s[i++]);
    putchar('\n');
}

```

17. 以下程序的功能是: _____。

```

#include <stdio.h>
#include <string.h>
void main()
{
    char str[10][80], c[80];
    int i;
    for(i=0;i<10; i++) gets(str[i]);
    strcpy(c, str[0]) ;
    for(i=1; i<10; i++)
        if(strlen(c)<strlen(str[i]))
            strcpy(c,str[i]);
    printf("%s\n",c);
}

```



```
    printf("%d\n",strlen (c));
}
```

18. 以下程序的功能是：_____。

```
#include <stdio.h>
#include <string.h>
void main()
{
    char a[10][80],c[80];
    int i,j,k;
    for(i=0; i<10; i++)
        gets(a[i]);
    for(i=0; i<9; i++)
    {
        k=i;
        for(j=i+1; j<10; j++)
            if(strcmp (a[j],a[k])<0)
                k=j;
        if(k!=i)
        {
            strcpy(c,a[i]); strcpy(a[i], a[k]); strcpy(a[k],c);
        }
    }
    for(i=0; i<10; i++)
        puts(a[i]);
}
```

19. 以下程序的输出结果为_____。

```
#include <stdio.h>
void main()
{
    int a[6]={12,4,17,25,27,16},b[6]={27,13,4,25,23,16},i,j;
    for(i=0;i<6;i++)
    {
        for(j=0;j<6;j++)
            if(a[i]==b[j])
                break;
        if(j<6)
            printf("%d",a[i]);
    }
    printf("\n");
}
```

20. 以下程序的输出结果为_____。

```
#include <stdio.h>
void main()
{
```

```

char a[8],temp; int j,k;
for(j=0;j<7;j++)
    a[j]='a'+j;
a[7]='\0';
for(j=0;j<3;j++)
{
    temp=a[6];
    for(k=6;k>0;k--)
        a[k]=a[k-1];
    a[0]=temp;
    printf("%s\n",a);
}

```

21. 以下程序的输出结果为_____。

```

#include <stdio.h>
#include <string.h>
void main()
{
    int i;
    char str1[ ]="*****";
    for(i=0;i<4;i++)
    {
        printf("%s\n",str1);
        str1[i]=' ';
        str1[strlen(str1)-1]='\0';
    }
}

```

22. 以下程序的输出结果为_____。

```

void main()
{
    float array[4][3]={ {3.4,-5.6,56.7}, {56.8,999.,-.0123}, {0.45,-5.77,123.5},
                        {43.4,0,111.2} };
    int i,j,min,m=0,n=0;
    min=array[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if(min>array[i][j])
            {
                min=array[i][j];
                m=i;n=j;
            }
    printf("min=%d,m=%d,n=%d\n",min,m,n);
}

```

23. 阅读以下程序并填空。

```
void main()
{
    char str[80];
    int i=0;
    gets(str);
    while(str[i]!='\0')
    {
        if(str[i]>='a'&&str[i]<='z')
            str[i]-=32;
        i++;
    }
    puts(str);
}
```

程序运行时如果输入 **upcase**，则屏幕显示_____。

程序运行时如果输入 **Aa1Bb2Cc3**，则屏幕显示_____。

三、判断题

1. C 语言允许动态定义数组大小，即可用变量定义数组大小。 ()
2. 字符数组不要求它的最后一个字符为 '\0'，甚至可以不包含 '\0'。 ()
3. 数组定义 “int a[10];” 也可以写成 “int a(10);”。 ()
4. 在对全部数组元素赋初值时，可以不指定数组长度。 ()
5. 定义 s 为 5 行 6 列的数组可写成 “float a[5,6];”。 ()
6. 数组定义 “int a[10];” 表示数组名为 a，此数组有 10 个元素。 ()
7. 数组定义 “int a[10];” 表示数组名为 a，此数组有 10 个元素，第 10 个元素为 a[10]。 ()
8. “static char c[]={"a book"};” 与 “static char c[]="a book";” 等价。 ()
9. “static char c[]={ 'a',' ','b','o','o','k','\0'};” 与 “static char c[]={ 'a',' ','b','o','o','k'};” 等价。 ()
10. 设已有说明 “static char c1[10],c2[10];”，则 “c1={"book"}; c2=c1;”。 ()

四、上机操作题

1. 编写程序，实现输入一串字符，将其中的英文字母加密、解密，非英文字母不变。
2. 有 n 个英文单词，编写程序，实现将其按字母顺序排列输出。
3. 编写程序，实现将用户输入的字符串以反向形式输出。例如，输入的字符串是 **abcdefg**，输出为 **gfedcba**。
4. 编写程序，实现从键盘读入一个字符串(该串在输入时以回车结束，且均为小写字母)，输出每个字母出现的次数。
5. 编写程序，实现从键盘输入一个字符串，要求在输入的字符串中每两个字符之间插入一个空格。例如，原串为 **aabbcc**，要求输出的新串为 **aa bb cc**。

6. 编写程序, 实现判断输入的一串字符是否为“回文”。“回文”是指顺序读和逆序读都一样的字符串。例如, “12321”和“abcdcba”都是回文。

7. 编写程序, 实现字符串循环右移 4 位。

8. 编写程序, 实现字符串中的大写字母变成对应的小写字母, 同时将其中的小写字母变成对应的大写字母, 其他字符不变。字符串由键盘读入。

9. 有一个字符串包含 n 个字符, 编写一个函数将此字符串中从第 m 个字符开始的全部字符复制成为另一个字符串。

10. 编写程序, 实现定义一个存储 10 个元素的整型数组, 从键盘上输入 10 个整数, 用选择排序法将它们按从大到小的顺序排列, 再从键盘输入一个整数, 用二分查找法查找该数是否在数组中。

11. 编写程序, 实现从键盘输入 9 个不同的整数, 组成 3 行 3 列的二维数组, 找出每一列中的最大元素, 并输出其行、列下标。

12. 编写程序, 实现将 30 位学生的学号和两门课程的成绩输入到一个二维数组中(数据均为整型数)。每位学生两门课程的总分也放在此数组中, 按学生总分的优劣打印出成绩单(成绩单中包括学号、两门课程成绩及总分)。

13. 编写程序, 实现从键盘输入一个字符串(不超过 80 个字符), 该串中有两个字符 x , 求两个字符 x 之间的子串, 打印出该子串及该子串的长度。

14. 编写程序, 实现将 1~255 之间任意一个十进制整数转换成二进制的数。

15. 编写程序, 实现从键盘上任意输入 10 个 1~99 之间的整数, 统计其个位数字, 是 0~9 的分别有多少。

16. 编写程序, 实现两个矩阵相乘。

17. 编写程序, 实现打印奇数阶“幻方”的程序。所谓“幻方”就是将 $1 \sim n^2$ 个连续的自然数放在一个 $n \times n$ 的方阵中, 使每行、每列, 以及两条对角线的和等于同一个数。例如, 三阶“幻方”可将自然数 1~9 变成如下方阵:

8	1	6
3	5	7
4	9	2

奇数阶“幻方”的构造方法是: 首先把 1 放在顶行正中间的位置, 然后把后继数按自然顺序放置在右上斜对角线上, 并且进行如下修改:

(1) 当到达顶行时, 下一个整数放在底行, 好像它在顶行的上面;

(2) 当到达右端时, 下一个整数放在左端列, 好像它是紧靠右端列的右方;

(3) 当到达的方格已经填上数字或到达右上角时, 下一个数就放在刚填写数的正下方。

18. 编写程序, 实现两个字符串连接起来置于第三个字符串中(不得使用 `strcat()` 函数)。

19. 编写程序, 实现从键盘输入两个字符串 $c1$ 和 $c2$, 若 $c1$ 比 $c2$ 长, 检查 $c2$ 是否是 $c1$ 的子串, 若 $c2$ 是 $c1$ 的子串, 则显示出两个串及 $c2$ 在 $c1$ 中的开始位置; 若 $c2$ 不是 $c1$ 的子串, 则显示出 NO。例如, $c1$ 为 shanghai, $c2$ 为 han, 显示结果为 shanghai han 2; $c1$ 为 shanghai, $c2$ 为 wuhan, 显示结果为 NO。

第7章 函 数

前面的章节中讲了 C 语言的基本单位是函数，一个 C 语言程序是由若干个功能函数构成的。为提高功能的独立性和代码的重用性，往往把程序中的某些功能模块化为一个函数，通过函数的调用，执行某个功能，这样可使程序的结构更清晰。本章针对函数的定义、调用及其他相关知识进行讲解。

学习目标

- 掌握函数的概念及相关定义
- 掌握函数的调用方法
- 理解形式参数(简称“形参”)与实际参数(简称“实参”)的关系
- 理解递归调用的含义
- 掌握局部变量与全局变量的作用域
- 学会使用文件包含功能
- 掌握宏定义的使用方法
- 了解条件编译指令的使用方法

7.1 函数的定义与调用

教师工资管理系统由 4 个模块组成，每个模块又分成若干个子功能，如修改记录功能、删除记录功能、按编号查询记录功能等。在执行程序时，首先要分别确定修改、删除、查询的教师信息是否存在，因此可以将此功能独立设计成一个功能函数，即源程序中的 `Locate()` 函数，其程序代码如下：

```
int Locate(int num,int cnt)    /*按教师编号查询教师信息*/
{
    int p=-1,i;
    for(i=0;i<cnt;i++)
    {
        if(tea[i].TeaNum==num)
        {
            p=i;                /*记录查询到的记录的下标*/
            break;
        }
    }
    return p;                  /*返回查询到的记录的下标，如果返回为-1，表示记录不存在*/
}
```

Locate() 函数与 printf() 函数不同, 需要由程序员自己定义后再使用, 而 printf() 可以直接使用, 称为库函数。

7.1.1 函数的分类

函数按定义者来分, 可以分为库函数和自定义函数。

到目前为止, 我们学习的 printf()、scanf() 函数等都是 ANSI C 标准定义的库函数。任何符合 ANSI C 的编译器, 无论它支持什么平台, 都必须提供这些库函数供用户使用。仅调用 ANSI C 库函数的程序, 具有很好的移植性, 可以在多种平台上编译运行。

用户可按自己意愿编写完成任意功能的函数, 称自定义函数, 如前面提到的 Locate() 函数。将相关函数集合到一起, 就可构成用户自己的函数库。本章重点讲解自定义函数。

7.1.2 函数的定义

C 语言中的函数与变量一样, 必须定义后才能使用。

函数定义的一般格式如下:

```
返回值类型  函数名(形参列表)          /*函数头*/
{
    变量声明
    函数实现语句
}
```

(1) 返回值类型

返回值类型是返回给主调函数的运算结果的数据类型。当返回值类型为 void 类型时, 说明函数没有返回值。函数的返回值通过函数中的返回语句 return 将被调函数中的一个确定的值带回到主调函数中。

return 语句的用法如下:

```
return(表达式);
```

或

```
return 表达式;
```

或

```
return;
```

例如:

```
return p;
```

或

```
return(p);
```

或

```
return(x>y?x:y);
```

如果需要从被调函数带回一个函数值(供主调函数使用),在被调函数中必须包含 `return` 语句。如果不需要从被调函数带回函数值,可以不要 `return` 语句。

一个函数中可以有一个以上的 `return` 语句,执行到哪个 `return` 语句,哪个就起作用。`return` 语句的功能如下。

① 使程序控制从被调函数返回主调函数,同时把返回值带给主调函数;释放在被调函数执行过程中分配的所有内存空间。

② 既然函数有返回值,这个值当然应属于某一个确定的类型,所以应在定义函数时需指定函数类型;凡不加类型说明的函数,一律自动按整型处理。

如果函数的类型和 `return` 语句中表达式的值不一致,则以函数类型为准。对于数值型数据,可以自动进行类型转换,即函数类型决定返回值的类型。

③ 可以使用 `void` 将函数定义为空类型或无类型,表示函数不带返回值。需要注意的是:`void` 类型的函数不是调用函数之后不再返回,而是被调函数在返回时没有返回值。`void` 类型在 C 语言中有两种用途:一是表示一个函数没有返回值,二是用来指明有关通用型的指针。

`void` 类型的函数和有返回值类型的函数在定义时没有区别,只是在调用时不同。有返回值的函数可以将函数调用放在表达式的中间,将返回值用于计算,而 `void` 类型的函数不能将函数调用放在表达式中,只能在语句中单独使用。

`void` 类型的函数多用于完成一些规定的操作,而主调函数本身不再对被调函数的执行结果进行引用。

④ 如果被调函数没有 `return` 语句,则函数没有返回值。

需要注意的是:如果在编写一个应该返回一个值的函数时忘记将这个值返回,将导致一个编译错误;如果从返回值类型是 `void` 的函数中返回一个值,也将导致一个编译错误。

(2) 函数名

函数名可以是任何合法的标识符,为了提高程序的可读性并减少注释,一般要求做到“见名知意”。

(3) 形参列表

形参列表是一组用逗号分隔的形式参数,它规定了函数被调用时应该接收到的参数,形参的命名最好也能使用有意义的名称。如果函数没有形参,那么圆括号中可以为空,也可以写 `void`。每个形参的前面都需要说明其数据类型,即使这几个参数具有相同的数据类型,也必须对每个形参分别进行类型说明。例如:

```
int max(int a,int b)
```

不能写成

```
int max(int a,b)
```

这将导致一个编译错误。

返回值类型、函数名和形参列表可称为函数头。

需要注意的是:函数头后面不能加分号,否则会产生语法错误;在函数体内,如果一个形参变量被再次定义成一个局部变量,将导致编译错误。

为了避免混淆,最好不让传递给函数的实参与这个函数的形参使用相同的变量名。

(4) 函数体

函数体内声明的变量专属于该函数，其他函数不能对这些变量进行操作。变量声明必须出现在所有可执行语句之前。包含在花括号内的变量声明和函数实现语句构成了函数体。

7.1.3 函数的调用

函数的调用是指在程序中使用已经定义过的函数，一般形式如下：

函数名(实参列表)；

说明：

① 调用函数时，函数的名称必须与具有该功能的自定义函数的名称完全一致。如果调用无参函数，则实参列表可以没有，但圆括号不能省略。

② 实参列表中的参数可以是常数、变量或表达式。如果实参不止一个，则相邻实参之间用逗号分隔。

③ 实参的个数、类型和顺序应该与被调函数要求的参数个数、类型和顺序一致，才能正确地进行数据传递。如果类型不匹配，C 编译程序将按赋值兼容的规则进行转换。如果实参和形参的类型不兼容，C 编译程序通常不会给出出错信息，程序仍然继续执行，但是得不到正确的结果。

④ 对实参列表求值的顺序并不是确定的，有的系统按自左向右的顺序求实参的值，有的系统则按自右向左的顺序执行。

在系统中，主函数判断用户的输入，选择调用对应的功能函数，并执行，其程序代码如下：

```
Menu(); /*调用菜单函数*/
printf("请输入你的选择: ");
scanf("%d",&choice);
switch(choice)
{
    case 1:
        count=Add(count);          /*调用添加记录函数*/
        break;
    case 2:
        ShowRecord(count);         /*调用显示记录函数*/
        break;
    case 3:
        Modify(count);             /*调用修改记录函数*/
        break;
    ...
    case 0:
        ExitSystem(count);         /*调用退出系统函数*/
    default:
        printf("选择错误, 按任意键返回主菜单");
}
```

函数的调用语句还可以出现在表达式中，例如：


```
n=3+max(a,b);
```

函数作为另一个函数调用的实参出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。例如：

```
n=max(a,max(b,c));
```

其中 `max(b,c)` 是一次函数调用，它的值作为 `max` 另一次调用的实参。`n` 的值是 `a`、`b`、`c` 三者的最大值。

又如：

```
printf("%d",max(a,b));
```

也是把 `max(a,b)` 作为 `printf()` 函数的一个参数。

函数调用作为函数的参数，实际也是函数表达式形式调用的一种，因为函数的参数本来就要求是表达式形式。

7.2 函数的参数传递

183

在调用函数时，大多数情况下主调函数和被调函数之间有数据传递关系，这就是前面提到的有参函数。在定义函数时，函数名后面圆括号中的参数为形参。在调用函数时，函数名后面圆括号中的参数为实参。

形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。形参和实参的功能是数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送。

在 C 语言中，实参向形参传送数据的方式是“值传递”。函数形参变量与实参变量的值的传递过程类似于日常生活中的“复印”操作：甲方请乙方工作，拿着原件为乙方复印了一份复印件，乙方凭复印件工作，将结果汇报给甲方。乙方在工作中可能会在复印件上进行涂改、增加、删除、注释等操作，但乙方对复印件的任何修改都不会影响到甲方的原件。

值传递的优点在于：被调函数不可能改变调用函数中变量的值，而只能改变它的局部的临时副本。这样就可以避免被调函数在操作时可能对主调函数中变量产生的影响。

【案例 7-1】 编写程序，实现调用函数时的数据传递。

```
#include <stdio.h>
int max(int x,int y)
/*定义有参函数max，x和y为形参，接收来自主调函数的原始数据*/
{
    int z;
    z=x>y?x:y;
    return(z);                /*将函数的结果返回主调函数*/
}

int main()
```

```

{
    int a,b,c;

    printf("input integer a,b:");
    scanf("%d,%d",&a,&b);
    c=max(a,b);          /*主函数内调用功能函数 max(), 实参为 a 和 b*/
    printf("max is %d\n",c);

    return 0;
}

```

```

input integer a,b:10,6
max is:10.
Press any key to continue...

```

图 7-1 运行结果

程序的运行结果如图 7-1 所示。

说明:

程序从主函数开始执行, 首先输入 a、b 的数值为 10、6, 然后调用函数 max(a,b)。具体的调用过程如下。

① 给形参 x、y 分配内存空间。

② 将实参 a 的值传递给形参 x, b 的值传递给形参 y, 于是 x 的值为 10, y 的值为 6。

③ 执行函数体, 给函数体内的变量分配存储空间, 即给 z 分配存储空间, 执行算法得到 z 的值为 10, 执行 return 语句, 完成以下功能: 将返回值返回主函数, 即将 z 的值返回给 main(); 释放函数调用过程中分配的所有内存空间, 即释放 x、y、z 的内存空间; 结束函数调用, 将流程控制权交给主调函数。

调用结束后, 继续执行 main() 函数直至结束。

注意:

① 函数定义中指定的形参变量在未出现函数调用时并不占用内存中的存储单元。只有在发生函数调用时, max() 函数中的形参才被分配内存单元。在调用结束后, 形参所占的内存单元被释放。

② 实参可以是常量、变量或表达式, 例如:

```
max(4, a+b);
```

但要求它们有确定的值。在调用时将实参的值赋给形参变量(如果形参是数组名, 则传递的是数组的首地址, 而不是数组的值)。

③ 实参与形参的类型相同或赋值兼容。案例 7-1 中实参和形参都是整型, 这是合法的、正确的。如果实参为整型而形参为实型, 或者相反, 则按前面介绍的不同类型数值的赋值规则进行转换。

例如: 实参为 6.6, 而形参为整型, 则先将实数 6.6 转换成整数 6, 然后再送到形参中。但此时应将 max() 函数放在 main() 函数的前面, 或在 main() 函数中对被调函数 max() 进行原型声名, 否则会出错。字符型与整型可以通用。

④ C 语言规定, 实参变量对形参变量的传递是“值传递”, 即单向传递, 只能由实参传给形参, 而不能由形参传回给实参。在内存中, 实参单元和形参单元是不同的单元, 如图 7-2 所示。

在调用函数时, 给形参分配存储单元, 并将实参的值传递给形参, 调用结束后, 形参单元被释放, 即形参 x、y 占用的存储单元被释放。实参单元仍保留并维持原值。

因此,在执行一个被调函数时,形参的值如果发生改变,并不会改变主调函数的实参的值。例如,若在执行函数过程中,x和y的值变为12和21,而a和b的值仍为10和6,如图7-3所示。

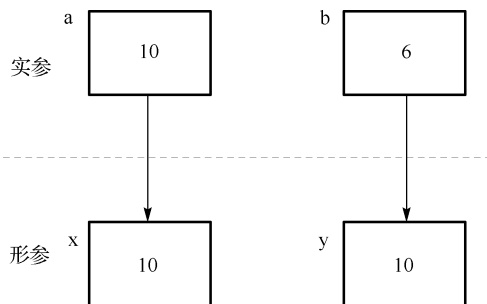


图 7-2 实参传值给形参

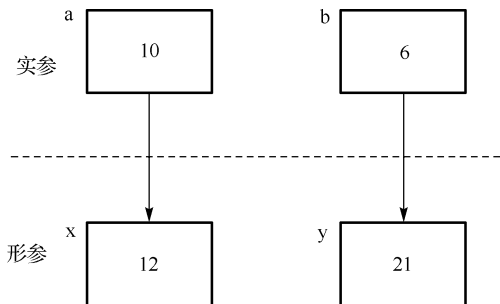


图 7-3 实参值不随形参值改变

【案例 7-2】 以下程序中 swap() 函数的功能是实现形参 x 与 y 值的交换,阅读并分析程序。

```
#include <stdio.h>
void swap(int x,int y)      /*定义函数 swap()*/
{
    int t;
    printf("swap before:x=%d,y=%d\n",x,y);
    t=x;x=y;y=t;
    printf("swap after:x=%d,y=%d\n",x,y);
    return;
}

int main()
{
    int x,y;                /*实参*/
    printf("input x,y:");
    scanf("x=%d,y=%d",&x,&y);
    swap(x,y);              /*调用函数*/
    printf("\n output :x=%d,y=%d\n",x,y);
    return 0;
}
```

程序的运行结果如图 7-4 所示。从运行结果看,swap() 函数实现了两个数的交换,当回到主函数中时,x和y的值没有发生变化。请分析原因,并思考如何实现两个数的交换。

```
input x,y: x=10,y=6
swap before:x=10,y=6
swap after:x=6,y=10
output :x=10,y=6
```

图 7-4 运行结果

7.3 函数的调用方式

在 C 语言中,除了 main() 函数不能被其他函数调用,其余的函数都能被调用。它们可以被 main() 函数调用,也可以被非 main() 函数调用。除此以外,函数还可以嵌套调用和递归调用。

7.3.1 函数的嵌套调用

当函数调用另一个函数时，这个被调函数中又调用了另一个函数，这种调用方式称为嵌套调用。例如，教师工资管理系统源程序，在 `main()` 函数中调用修改记录 `Modify()` 函数的语句如下：

```
case 3:
    Modify(count);          /*调用修改记录函数*/
    break;
```

修改记录 `Modify()` 函数中又调用查询修改教师信息是否存在的 `Locate()` 函数的语句如下：

```
scanf("%d",&tempnum);
p=Locate(tempnum,cnt); /*调用按编号查询教师信息函数*/
```

`main()` 函数调用 `Modify()` 函数，`Modify()` 函数又调用 `Locate()` 函数，如此就形成了函数的嵌套调用。需要注意的是，函数可以嵌套调用，但不能嵌套定义，即一个函数不能定义在另一个函数体内。

7.3.2 函数的递归调用

当函数调用另一个函数时，这个被调函数中又调用了另一个函数，这个函数不是其他的函数，而是它本身，这种情况在 C 语言中是允许出现的，称为递归调用。自己调用自己的函数称为递归函数。递归函数往往是把一个大型复杂的问题层层转换为一个与原问题相似，但规模较小的问题来求解。递归只需要少量代码就可描述出解题过程所需要的多次重复计算，大大减少了程序的代码量。

【案例 7-3】 利用 $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$ ，计算整数 n 的阶乘 ($n!$)。

先采用迭代法编程计算 $n!$ ，程序代码如下：

```
#include <stdio.h>
int main(void)
{
    int n,i;
    long result=1;
    printf("Input n: ");
    scanf("%d",&n);
    for (i=1;i<=n;i++)
    {
        result*=i;          /*计算 n!*/
    }
    if (n>=0)
    {
        printf("%d!=%ld\n",n,result);
    }
    return 0;
}
```

程序采用迭代法，每个 $i!$ 的结果都是由前一个 `result` 乘以现在的 i 得到的，当 i 循环到等于 n 的时候，`result` 就是 $n!$ 的结果。事实上，也可以用 $(n-1)!$ 来计算 $n!$ ，即 $n!=n*(n-1)!$ 。同理再用 $(n-2)!$ 来计算 $(n-1)!$ ，即 $(n-1)!=(n-1)*(n-2)!$ 。其余以此类推，直到用 $2!=2*1!$ 和 $1!=1$ 计算出 $2!=2$ 时为止，这种递推关系也可用如下递归公式表示：

$$n!=\begin{cases} 1 & n=0, 1 \\ n \times (n-1)! & n \geq 2 \end{cases}$$

用递归的方法实现计算阶乘函数的程序代码如下：

```
#include <stdio.h>
long fact(long n);
int main(void)
{
    int n;
    long result;
    printf("Input n: ");
    scanf("%d", &n);
    result=fact(n);
    if (result!=0)
    {
        printf("%d!=%ld\n", n, result);
    }
}
/*函数功能：当n大于2时，递归计算n!的值。当n为0或1时，返回1；当n小于0时，返回0*/
long fact(long n)
{
    if (n<0)
    {
        printf("data error!\n");
        return 0;                /*如果n小于0，则返回0*/
    }
    else if (n==0||n==1)        /*递归终止条件*/
    {
        return 1;                /*当n为0或1时，返回1*/
    }
    else
    {
        return n*fact(n-1);      /*递归调用计算n!*/
    }
}
```

两个程序的运行结果完全一样，如图 7-5 所示。

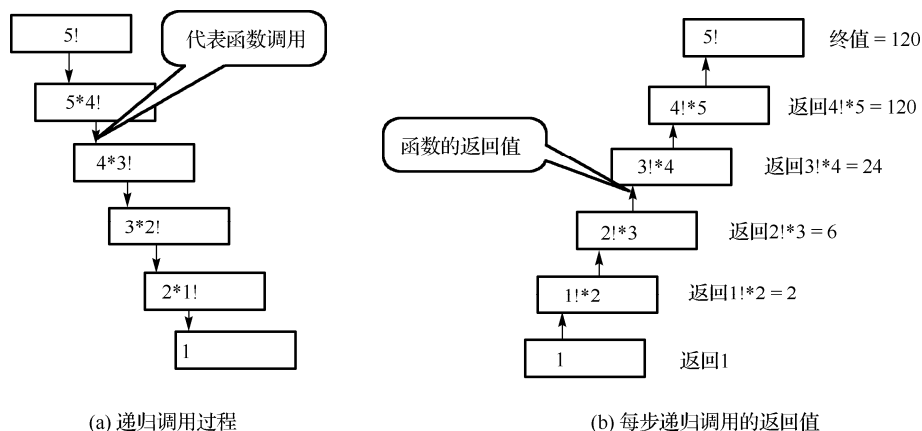
结果虽然一样，但两个程序的运行过程截然不同， $5!$ 的递归调用过程如图 7-6 所示。

主要步骤如下。

① 为了计算 $5!$ ，主程序调用 `fact(5)` 计算 $5!$ ，但 `fact(5)` 并没有直接计算 $5!$ ，而是在 `fact(5)` 中完成 $5*\text{fact}(4)$ 的计算。

Input n: 5
5! = 120

图 7-5 运行结果

图 7-6 $5!$ 的递归调用过程

② 在 $\text{fact}(4)$ 中计算 $4!$ ，但 $\text{fact}(4)$ 没有直接计算 $4!$ ，而是在 $\text{fact}(4)$ 中完成 $4*\text{fact}(3)$ 的计算。

③ 在 $\text{fact}(3)$ 中计算 $3!$ ，但 $\text{fact}(3)$ 没有直接计算 $3!$ ，而是在 $\text{fact}(3)$ 中完成 $3*\text{fact}(2)$ 的计算。

④ 在 $\text{fact}(2)$ 中计算 $2!$ ，但 $\text{fact}(2)$ 没有直接计算 $2!$ ，而是在 $\text{fact}(2)$ 中完成 $2*\text{fact}(1)$ 的计算。

⑤ $\text{fact}(1)$ 的入口参数为 1， $\text{fact}(1)$ 中判断递归终止的条件成立，则 $\text{fact}(1)$ 返回整数 1 给调用它的函数 $\text{fact}(2)$ ，同时退出 $\text{fact}(1)$ ，返回 $\text{fact}(2)$ 。

⑥ 在 $\text{fact}(2)$ 中，计算 $\text{fact}(1)$ 的返回值与 2 的乘积，并将 $2*\text{fact}(1)$ 的计算结果返回给调用 $\text{fact}(2)$ 的函数 $\text{fact}(3)$ ，同时退出 $\text{fact}(2)$ ，返回 $\text{fact}(3)$ 。

⑦ 在 $\text{fact}(3)$ 中，计算 $\text{fact}(2)$ 的返回值与 3 的乘积，并将 $3*\text{fact}(2)$ 的计算结果返回给调用 $\text{fact}(3)$ 的函数 $\text{fact}(4)$ ，同时退出 $\text{fact}(3)$ ，返回 $\text{fact}(4)$ 。

⑧ 在 $\text{fact}(4)$ 中，计算 $\text{fact}(3)$ 的返回值与 4 的乘积，并将 $4*\text{fact}(3)$ 的计算结果返回给调用 $\text{fact}(4)$ 的函数 $\text{fact}(5)$ ，同时退出 $\text{fact}(4)$ ，返回 $\text{fact}(5)$ 。

⑨ 在 $\text{fact}(5)$ 中，计算 $\text{fact}(4)$ 的返回值与 5 的乘积，并将 $5*\text{fact}(4)$ 的计算结果返回给调用 $\text{fact}(5)$ 的主函数 $\text{main}()$ ，同时退出 $\text{fact}(5)$ ，返回 $\text{main}()$ 。

⑩ 在 $\text{main}()$ 中打印 $5!$ 的结果。

在函数递归调用时，需要确定两点：一是递归公式，二是边界条件。递归公式是递归求解过程中的归纳项，边界条件即终止条件，用于终止递归。

【案例 7-4】 编写程序，实现猴子吃桃问题的计算。设有若干个桃子，1 只猴子第 1 天吃了所有桃子的一半多 1 个，第 2 天吃了剩下桃子的一半多 1 个，每天如此，到第 7 天吃时只有 1 个桃子了，求一共有多少个桃子？

案例分析：设程序中 $\text{cal}(n)$ 表示第 n 天原有的桃子数。第 7 天吃时只有 1 个桃子了，则 $\text{cal}(7)=1$ 。求共有多少个桃子，就是求第 1 天原有多少个桃子，即求 $\text{cal}(1)$ 是多少。如果知道 $\text{cal}(2)$ 的值（第 2 天原有的桃子数也是第 1 天剩下的桃子数）就好了，因为有 $\text{cal}(1)=(\text{cal}(2)+1)*2$ （由 $\text{cal}(2)=\text{cal}(1)-(\text{cal}(1)/2+1)$ 可得），这样求 $\text{cal}(1)$ 就变成了求 $\text{cal}(2)$ 。问题的性质相同，但规模变小了。猴子吃桃的规律相同，所以函数 $\text{cal}(n)$ 可定义如下：

当 $n=7$ 时， $\text{cal}(n)=1$ ；

当 $n=1\sim6$ 时， $\text{cal}(n)=(\text{cal}(n+1)+1)*2$ 。

根据递归公式和边界条件，编写递归函数，程序代码如下：

```
#include<stdio.h>
int cal(int n)
{
    if(n==7)
        return 1;
    else
        return (cal(n+1)+1)*2;
}
int main()
{
    int s;
    s=cal(1);
    printf("共有桃子%d个.\n",s);
    return 0;
}
```

7.4 变量的作用域

标识符的声明是为编译器提供有关该标识符含义的信息。声明变量时，变量除了数据类型这一属性，还有很多其他属性，如存储类型、作用域和链接等。

7.4.1 变量的存储类型

变量的存储类型决定了它的存储周期、作用域和链接。变量的存储周期是指变量在内存中存在的时间。有些变量的存在时间很短，有些变量反复被创建、收回，有些变量则在程序的整个运行期间都留在内存中。变量的作用域是指变量在程序中能够被访问到的区域。有些变量在程序的任何地方都能被访问，而有些变量只能在程序的一部分地方被访问。变量的链接属性是针对由多个源文件组成的程序而言的，旨在说明这个变量能否被其他源文件访问。

变量的存储周期分为自动存储周期和静态存储周期。具有自动存储周期的变量的存储空间在执行到它所在块时才被创建，在退出程序块时被释放。具有静态存储周期的变量的存储空间在整个程序运行期间都存在，并占用同一存储空间。

变量的存储类型决定了为变量分配内存和释放内存的时间。变量的存储类型有 4 种，分别为：**auto**(自动类型)、**static**(静态类型)、**extern**(外部类型)和 **register**(寄存器类型)。

1. auto 存储类型

auto 存储类型的变量只在其所在块内有效，在所在块内执行时获得内存单元，并在块终止时释放内存单元。例如：

```
auto int n;
```

该语句声明了 **n** 是具有自动存储周期的局部变量。**auto** 存储类型几乎不用明确指出，因为函数的局部变量(在函数体内声明的变量和函数形参列表中的变量)均默认为 **auto** 存储类型。将具有自动存储周期的变量简称为“自动变量”。

自动存储是一种节约内存的手段，并且符合“最小权限原则”，因为自动变量只在需要它们的时候才占用内存。

2. static 存储类型

具有静态存储周期的变量占用的存储单元是从程序运行开始时分配和初始化的，并且只分配和初始化一次。但这并不意味着在程序的任何地方都可以访问它们。存储周期和作用域是两个不同的概念。

可以将局部变量声明为 **static** 存储类型，称为“静态局部变量”。它在整个程序执行期间拥有永久的存储单元，一直会保留变量的值。

外部变量具有静态存储周期。将一个变量定义为外部变量的方法是将其声明在任何一个函数体之外，外部变量的存储空间在整个程序运行期间始终存在，但只可以被位于其声明语句之后的函数访问。

3. extern 存储类型

extern 存储类型可以使几个源文件共享一个变量，提醒编译器需要访问定义在别处的变量，可能是同一文件稍后的位置，也可能在另一个文件中。注意，**extern** 不是变量定义，只是声明一个定义在别处的变量。

4. register 存储类型

当程序执行时，数据通常存储在内存中。当需要计算或进行其他处理时，数据才被装入 CPU 的寄存器。声明变量为 **register** 类型就要求编译器把变量存储在寄存器中。寄存器是驻留在 CPU 中的存储单元，具有比内存更高的存取速度。一般将诸如循环变量或累加求和变量等需要频繁访问的变量声明为寄存器类型。例如：

```
register int sum;
```

事实上，**register** 声明常常是多余的，因为现在的编译器具有很好的优化能力，它会找出频繁访问的变量，并将其驻留在寄存器中，而无须 **register** 声明。

7.4.2 全局变量与局部变量

形参变量只在被调用期间才分配内存单元，调用结束立即释放。这一点表明形参变量只有在函数内才是有效的，离开该函数就不能再使用了。这种表示变量有效性的范围称为变量的作用域，也称为变量的可见性。不仅对于形参变量，C 语言中所有的变量都有自己的作用域。变量说明的方式不同，其作用域也不同。C 语言中的变量按作用域范围可分为两种，即局部变量和全局变量。

1. 局部变量

在一个函数或复合语句内定义的变量称为局部变量，也称为内部变量。局部变量仅在定义它的函数或复合语句内有效。例如，函数的形参是局部变量。

编译时，编译系统不为局部变量分配内存单元，而是在程序的运行中，当局部变量所在的函数被调用时，编译系统根据需要临时分配内存。当函数调用结束时，局部变量的空间被释放。

在教师工资管理系统中，每个函数都有相应的局部变量，部分函数的局部变量如下：

```
int Add(int cnt)           /*添加教师记录*/
{
    int choice;
    int i,teaNumTemp,j,recordcnt;
    .../*省略功能语句*/
    return recordcnt;      /*返回记录数*/
}
void ShowRecord(int cnt)   /*显示所有记录*/
{
    int i,p;
    .../*省略功能语句*/
}
void Modify(int cnt)       /*修改教师工资信息*/
{
    int tempnum,p,teaNumTemp,j;
    .../*省略功能语句*/
}
void main()
{
    int choice=0,count=0;
    .../*省略功能语句*/
}
```

在 `Add()`、`ShowRecord()` 和 `Modify()` 函数中都有一个同名的形参 `cnt`，但这 3 个形参只在相应的函数范围内有效，函数调用结束，`cnt` 变量的空间会被立即释放。同理，在 `Add()` 和 `main()` 函数中有 `choice` 局部变量，其有效范围也仅在相应的函数内。在局部变量的使用过程中需要注意以下几点。

形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。

允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰。例如，形参和实参的变量名都为 `n`，是完全允许的。

在复合语句中也可定义变量，其作用域只在该复合语句范围内，离开该复合语句该变量无效，将释放内存单元。

2. 全局变量

全局变量也称外部变量，是在函数外部定义的变量。其不属于哪一个具体函数，而属于一个源程序文件。作用域是从定义的位置到本文件的结尾处。局部变量在定义时不会自动初始化，除非指定初值。全局变量在不指定初值的情况下自动初始化为零。在教师工资管理系统中，`saveflag` 变量是一个全局变量，由于定义在所有函数之前，所以它在所有函数中都有效。

在一个函数中既可以使用本函数中的局部变量，又可以使用有效的全局变量。在同一个源文件中，允许全局变量和局部变量同名，但在局部变量的作用域内，全局变量被“屏蔽”，不起作用。

【案例 7-5】 程序示例如下。

```
#include<stdio.h>
int max=10;          /*全局变量 max*/
int Max(int x,int y)
{
    int max=0;        /*局部变量 max，全局变量 max 在此处不起作用*/
    max=x>y?x:y;
    return max;
}
int main()
{
    int max;           /*局部变量 max，全局变量 max 在此处也不起作用*/
    max=Max(100,200);
    printf("%d",max);
    return 0;
}
```

在此程序中，包含全局变量名和局部变量名，都为 `max`。在 `Max()` 函数和 `main()` 函数中的局部变量 `max` 可起作用，而值为 10 的全局变量 `max` 在有同名局部变量的作用域内被“屏蔽”，将不起作用，所以此程序的输出结果为 200。

192

7.5 编译预处理

7.5.1 文件包含

文件包含是指一个源文件可以将另一个源文件的全部内容包含进来，即将另外的文件包含到本文件中。C 语言提供了 `#include` 命令，用来实现文件包含的操作。文件包含命令的一般形式如下：

```
#include<包含文件名>
```

或

```
#include"包含文件名"
```

这两种格式都能够实现文件包含，不同的是：第 1 种格式是标准格式，当使用这种格式时，C 编译系统将在系统指定的路径下搜索相应的文件。常见的文件包含语句如下：

```
#include<stdio.h>
```

`stdio.h` 头文件在 VC++ 6.0 安装路径的“VC98\include”文件夹中。

当使用第 2 种格式时，系统首先会在用户当前工作的目录中搜索文件，如果找不到，再按系统指定的路径进行搜索。

文件包含命令的功能是，把指定的文件插入该命令行位置，取代该命令行，从而把指定的文件和当前的源程序文件连接成一个源文件。其实 `scanf()`、`printf()` 等基本输入、输出函数

都被定义到 `stdio.h` 文件中, 所以源程序在使用这些函数之前都会把 `stdio.h` 文件包含进来。如果要使用 `sqrt()` 函数, 就需要将 `math.h` 文件包含进来。

在程序设计中, 文件包含功能是很有用的。一个复杂的程序可以分为多个模块, 由多个程序员分别编程。有些公用的符号常量或宏定义等可单独组成一个包含文件(也称为头文件, 常以“.h”为后缀), 在其他文件的开头用包含命令包含该文件后即可使用。这样, 可避免在每个文件的开头重复书写公用量, 以节省时间, 并减小出错率。

7.5.2 宏定义与替换

C 语言中宏用 `define` 命令定义, 一般形式如下:

```
#define 标识符 值
```

其中, “标识符”称为宏名, “值”称为宏体。宏定义后, 源文件中出现的宏体就可以用宏名代替了。在程序中使用宏又称为宏引用。预处理时以标识符形式出现的宏名会替换成宏体, 这个过程称为“宏展开”。宏展开只是简单的查找替换操作。

当一个值(如 3.1415926)会多次出现在程序中且比较复杂时, 可以将这个值定义为宏, 语句如下:

```
#define PI 3.1415926
```

一个值在程序中多次出现, 且在编程期间可能会变化, 可以将其定义为宏。在程序中用宏名替换该值, 当值改变时, 只需修改宏定义, 不必在程序中逐个修改值。通常, 将数组的长度定义为宏就属于这种情况。

宏可分为两类: 简单宏和带参宏。

(1) 简单宏(无参宏)

用一个指定的标识符(即名称)来代表一个字符串, 其一般形式如下:

```
#define 标识符 字符串
```

其中, `define` 为宏定义命令, “标识符”为定义的宏名, “字符串”可以是常数、表达式、格式串等。

宏定义的功能是在进行编译前, 用字符串原样替换程序中的标识符。

【案例 7-6】 编写程序, 实现输入半径, 求圆的周长、面积, 以及球体的体积。要求使用无参宏定义圆周率。

```
#include <stdio.h>
#define PI 3.1415926          /*PI 是宏名, 3.1415926 用来替换宏名的常数*/
int main()
{
    double radius, length, area, volume;
    printf ("Input a radius: ");
    scanf ("%lf", &radius);
    length=2*PI*radius;        /*引用无参宏求圆的周长*/
    area=PI*radius*radius;     /*引用无参宏求圆的面积*/
    volume=PI*radius*radius*radius*3/4; /*引用无参宏求球体的体积*/
}
```

```
printf("length=%.2lf,area=%.2lf,volume=%.2lf\n", length, area, volume);
return 0;
}
```

程序的运行结果如图 7-7 所示。程序中 3.1415926 这个值如果发生变化,可以通过只修改宏定义来实现全部修改,即“一改全改”。

```
Input a radius: 5
length=31.42,area=78.54,volume=294.52
```

图 7-7 运行结果

(2) 带参宏

C 语言允许宏带有参数。在宏定义中的参数称为形参,在宏调用中的参数称为实参。对带参数的宏,在调用中,不可仅进行简单的字符串替换,还要进行参数替换。即不仅要宏展开,还要用实参去替换形参。

带参宏定义的一般形式如下:

```
#define 宏名(形参表) 字符串
```

其中,“字符串”中包含圆括号中指定的参数。

带参宏调用的一般形式如下:

```
宏名(实参表);
```

例如:

```
#define M(x) x*x+3*x /*宏定义*/
k=M(2); /*宏调用*/
```

在宏调用时,用实参 2 去替换形参 x,经预处理宏展开后的语句为:

```
k=2*2+3*2;
```

【案例 7-7】带参宏程序示例。

```
#include<stdio.h>
#define N 3+1
#define SQUARE(x) x*x
int main()
{
    printf("SQUARE(3)=%d\n",SQUARE(3));
    printf("SQUARE(N)=%d\n",SQUARE(N));
}
```

宏引用 SQUARE(3)展开后为 3*3,值为 9。宏引用 SQUARE(N)展开后为 N*N,再次展开后为 3+1*3+1,值为 7。

宏定义的作用范围是从宏定义命令开始到程序结束。如果需要在源程序的某处终止宏定义,则需要使用#undef 命令取消宏定义。取消宏定义命令#undef 的用法如下:

```
#undef 标识符
```

其中,标识符指定宏名。

7.5.3 条件编译

一般情况下，源程序中所有的行都参加编译。但如果用户希望某一部分程序在满足某条件时才进行编译，否则不编译，或按条件编译另一组程序，这时就要用到条件编译。预处理程序提供条件编译的功能，可以按不同的条件编译不同部分的程序，因此产生不同的目标代码文件。这对于程序的移植和调试是很有用的。

进行条件编译的宏指令主要有：`#if`、`#ifdef`、`#ifndef`、`#endif`、`#else` 等。它们按照一定的方式组合，构成条件编译的程序结构。

(1) 第 1 种形式

```
#ifdef 标识符
    程序段 1
#else
    程序段 2
#endif
```

功能是：如果标识符已被`#define` 命令定义，则对程序段 1 进行编译；否则对程序段 2 进行编译。如果没有程序段 2(为`空`)，`#else` 可以省略，则可以改写为

```
#ifdef 标识符
    程序段
#endif
```

其中“程序段”可以是语句组，也可以是命令行。

(2) 第 2 种形式

```
#ifndef 标识符
    程序段 1
#else
    程序段 2
#endif
```

功能是：如果标识符未被`#define` 命令定义，则对程序段 1 进行编译，否则对程序段 2 进行编译。这与第 1 种形式的功能正相反。

(3) 第 3 种形式

```
#if 常量表达式
    程序段 1
#else
    程序段 2
#endif
```

功能是：如果常量表达式的值为真(非 0)，则对程序段 1 进行编译，否则对程序段 2 进行编译。因此，可以事先给定条件，使程序在不同的条件下，完成不同的功能。

7.6 本章小结

本章主要讲解函数的基本定义，函数调用时的数据传递，变量的作用域，函数调用方式

等函数知识，以及编译预处理的三种方式。通过本章的学习，读者应掌握函数的定义方法，函数的调用方式，包括嵌套调用和递归调用，理解宏定义与替换，并能将相关知识运用到实际的编程中。

7.7 习题

一、单选题

1. C 语言总是从()函数开始执行。
A. main B. 处于最前的 C. 处于最后的 D. 随机选一个
2. 函数在定义时，省略函数类型说明符，则该函数值的类型为()。
A. int B. float C. long D. double
3. 在 C 言中，有关函数的说法，正确的是()。
A. 函数可嵌套定义，也可嵌套调用 B. 函数可嵌套定义，但不可嵌套调用
C. 函数不可嵌套定义，但可嵌套调用 D. 函数不可嵌套定义，也不可嵌套调用
4. 以下函数调用语句中，含有实参的个数为()。

```
fun((2,3),(4,5+6,7));
```

- A. 1 B. 2 C. 5 D. 6
5. 函数调用可以在()。
A. 函数表达式中 B. 函数语句中 C. 函数参数中 D. 以上都是
6. 被调函数返回给主调函数的值称为()。
A. 形参 B. 实参 C. 返回值 D. 参数
7. ()，可以不进行函数类型声明。
A. 被调函数的返回值是整型或字符型时
B. 被调函数的定义在主调函数定义之前时
C. 在所有函数定义前，已在函数外预先说明了被调函数类型
D. 以上都是
8. 被调函数通过()语句，将值返回给主调函数。
A. if B. for C. while D. return
9. 被调函数调用结束后，返回到()。
A. 主调函数中该被调函数的调用语句处
B. 主函数中该被调函数的调用语句处
C. 主调函数中该被调函数调用语句的前一语句
D. 主调函数中该被调函数调用语句的后一语句
10. 以下对 C 语言函数的有关描述，正确的是()。
A. 在 C 语言中，调用函数时，只能把实参的值传送给形参，形参的值不能传送给实参
B. C 语言函数既可以嵌套定义又可以递归调用

- C. C 语言函数必须有返回值, 否则不能使用函数
 - D. C 语言程序中有调用关系的所有函数必须放在同一个源程序文件中
11. C 语言中函数的隐含存储类型是()。
 - A. auto
 - B. static
 - C. extern
 - D. 无存储类型
12. 以下方法不能把函数处理结果的两个数据返回给主调函数的是()。
 - A. return 这两个数
 - B. 形参用两个元素的数组
 - C. 形参用两个这种数据类型的指针
 - D. 用两个全局变量
13. C 语言可执行程序从什么地方开始()。
 - A. 程序中第 1 条可执行语句
 - B. 程序中第 1 个函数
 - C. 程序中的 main() 函数
 - D. 包含文件中的第 1 个函数
14. 有一个函数原型如下所示, 则该函数的返回值类型为()。

```
abc(float x, float y);
```

- A. void B. double C. int D. float
15. 以下关于文件包含的说法中错误的是()。
- A. 文件包含指一个源文件可以将另一个源文件的全部内容包含进来
- B. 文件包含处理命令的格式为`#include`"包含文件名"或`#include`<包含文件名>
- C. 一条包含命令可以指定多个被包含文件
- D. 文件包含可以嵌套, 即被包含文件中又包含另一个文件
16. 以下程序的输出结果是()。

```
#define MAX(x,y) (x)>(y)?(x):(y)
#include <stdio.h>
int main()
{
    int a=5,b=2,c=3,d=3,t;
    t=MAX(a+b,c+d)*10;
    printf("%d\n",t);
    return 0;
}
```

17. 以下程序的功能是, 通过带参宏定义求圆的面积, 空格处语句应为()。

```
#define PI 3.1415926
#define AREA(r) _____
#include <stdio.h>
int main()
{
    float r=5;
    printf("%f",AREA(r));
    return 0;
}
```

- A. $\text{PI}^*(\text{r})^*(\text{r})$ B. $\text{PI}^*(\text{r})$ C. r^*r D. $\text{PI}^*\text{r}^*\text{r}$

18. 以下叙述正确的是()。
- A. 可以把 `define` 和 `if` 定义为用户标识符
 - B. 可以把 `define` 定义为用户标识符, 但不能把 `if` 定义为用户标识符
 - C. 可以把 `if` 定义为用户标识符, 但不能把 `define` 定义为用户标识符
 - D. `define` 和 `if` 都不能定义为用户标识符
19. 以下程序的运行结果为()。

```
#define PI 3.14
#define R 5.0
#define S PI*R*R
#include <stdio.h>
int main()
{
    printf("%f",S);
    return 0;
}
```

- A. 3.14 B. 78.500000 C. 5.0 D. 无结果

20. 以下函数调用语句中含有的参数个数是()。

```
func((a,b),(c,d,e),f);
```

- A. 3 B. 4 C. 5 D. 6

21. 以下程序的输出结果为()。

```
#include<stdio.h>
#define MIN(x,y) (x)<(y)?(x):(y)
void main(void)
{
    int i,j,k;
    i=5;j=10;
    k=10*MIN(i,j);
    printf("%d\n",k);
}
```

- A. 100 B. 50 C. 10 D. 5

22. 以下程序的功能是()。

```
#include <stdio.h>
void reverse()
{
    char c;
    if((c=getchar())!='\n')
        reverse();
    if(c!='\n')
        putchar(c);
}
void main(void)
{
```



```
reverse();  
printf("\n");  
}
```

- A. 顺序输出从键盘输入的字符
B. 逆序输入从键盘输入的字符
C. 顺序输出从键盘输入的字符的前半部分
D. 顺序输出从键盘输入的字符的后半部分
23. 以下说法正确的是()。
- A. 用户若需调用标准库函数, 调用前必须重新定义
C. 系统根本不允许用户重新定义标准库函数
B. 用户可以重新定义标准库函数, 但该函数失去原有含义
D. 用户若需调用标准库函数, 调用前不必使用预编译命令将该函数所在文件包括到用户源文件中, 系统会自动调用
24. 以下函数首部的正确定义形式是()。
- A. double fun(int x,int y) B. double fun(int x;int y)
C. double fun(int x,int y); D. double fun(int x,y);
25. 以下说法正确的是()。
- A. 实参与其对应的形参分别占用独立的存储单元
B. 实参与其对应的形参共同占用一个存储单元
C. 只有当实参与其对应的形参同名时才占用共同的存储单元
D. 形参是虚拟的, 不占用存储单元
26. 若调用一个函数, 且此函数中没有 return 语句, 则该函数()。
- A. 没有返回值 B. 返回若干个系统默认值
C. 能返回一个用户希望的函数值 D. 返回一个确定的值
27. C 语言规定, 简单变量作为实参时, 它和对应形参之间的数据传递方式是()。
- A. 地址传递
B. 单向值传递
C. 由实参传给形参, 再由形参传回给实参
D. 由用户指定传递方式
28. C 语言规定, 函数返回值的类型是由()。
- A. return 语句中的表达式类型决定的
B. 调用该函数时的主调函数类型决定的
C. 调用该函数时系统临时决定的
D. 在定义该函数时指定的函数类型决定的
29. 以下说法不正确的是()。
- A. 在不同函数中可以使用相同名字的变量
B. 形式参数是局部变量
C. 在函数内定义的变量只在函数范围内有效
D. 在函数内的复合语句中定义的变量在本函数范围内有效

30. 以下程序的运行结果是()。

```
#include <stdio.h>
void main(void)
{
    int fun(int a);
    int a=2,i,k;
    for(i=0;i<2;i++)
    {
        k=fun(++a);
        printf("%d",k);
    }
}
int fun(int a)
{
    int b=0;
    static int c=10;
    b++;
    a=a+b+c++;
    return a;
}
```

A. 1315 B. 1415 C. 1416 D. 1516

31. 如果把第 30 题程序中的 `static` 去掉, 则程序的运行结果是()。

A. 1315 B. 1415 C. 1416 D. 1516

32. 以下程序在(1)处调用 `trans()` 函数的输出结果是()。

```
#include <stdio.h>
void trans(int m,int n)
{
    int i;
    if (m>=n)
    {
        i=m%n;
        trans(m/n,n);
    }
    else
        i=m;
    if (i<10)
        printf("%d",i);
    else
        printf("%c",'A'+i-10);
}
void main( void )
{
    int m,n;
    printf("\nThe output is:\n\n");
    printf("\n%d->%d=",123,16);
}
```

```

    trans(123,16);          /*(1)*/
    printf("\n");
    scanf("%d,%d",&m,&n);
    printf("\n%d->%d=",m,n);
    trans(m,n);             /*(2)*/
}

```

- A. 7B B. 711 C. B7 D. 117
33. 若运行第 32 题程序,从键盘输入 65 和 5,则程序在 (2) 处调用 `trans()` 函数的输出结果是()。
- A. 32 B. 032 C. 23 D. 230
34. 以下函数,没有返回值的是()。
- A. `int a() {int a=2;return (a);}` B. `void b() {printf("c");}`
 C. `int a() {int a=2;return a;}` D. 以上都是
35. 以下关于预处理的叙述中不正确的是()。
- A. C 语言源程序中凡是以“#”开始的控制行都是预处理命令行
 B. 预处理命令行必须位于源程序的开始处
 C. 一条有效的预处理命令行必须单独占一行
 D. 预处理命令是在正式编译之前被处理的
36. 以下关于文件包含的描述正确的是()。
- A. 每个 C 语言程序必须包含预处理命令 `#include <stdio.h>`
 B. `#include` 后面的文件名用双引号(" ")括起来和用尖括号(< >)括起来完全等效
 C. `#include` 命令行可以出现在源程序中需要的任何地方
 D. 用 `#include` 包含的文件称为头文件,必须以 .h 作为扩展名
37. 以下有关宏替换的叙述不正确的是()。
- A. 宏名不具有类型
 B. 宏名必须用大写字母表示
 C. 宏替换只是字符替换
 D. 宏替换不占用运行时间
38. C 语言的编译系统对宏命令的处理是()。
- A. 在对源程序中其他成分正式编译之前进行的
 B. 和 C 语言程序中的其他语句同时进行的
 C. 在程序链接时进行的
 D. 在程序运行时进行的

二、填空题

- 变量的作用域主要取决于变量的_____, 变量的生存期既取决于变量的_____, 又取决于变量的_____。
- 说明变量时,若省略存储类型符,系统默认其为_____存储类型,该存储类型的类型符为_____。
- 静态型局部变量的作用域是_____, 生存期是_____。

4. 函数中的形参和调用时的实参都是数组名时, 传递方式为_____, 都是变量时, 传递方式为_____。

5. 函数的形式参数的作用域为_____, 全局的外部变量和函数体内定义的局部变量重名时, _____变量优先。

6. 若自定义函数要求返回一个值, 则应在该函数体中有一条_____语句, 若自定义函数要求不返回一个值, 则应在该函数说明时加一个类型说明符_____。

7. 若函数的形式参数是指针类型, 则实参可以是_____或_____。

8. 函数的参数为 `char *` 类型时, 形参与实参结合的传递方式为_____。

9. 函数的实参为常量时, 形参与实参结合的传递方式为_____。

三、判断题

1. 函数说明指在程序中设定一个函数模块。 ()

2. 形参只有在被调用时才分配存储空间。 ()

3. 在 C 语言函数中, 最好使用全局变量。 ()

4. 在调用函数时, 实参值传给形参, 调用结束时, 形参值传给实参。 ()

5. 所有函数定义都是并行的, 是相互独立的。 ()

6. 函数的隐含存储类型是 `extern`。 ()

7. 形参可以是常量、变量或表达式。 ()

8. 函数调用可以作为一个函数的形参。 ()

9. C 语言规定, 实参应与其对应的形参类型一致。 ()

10. 定义函数时, 形参的类型说明可以放在函数体内。 ()

四、上机操作题

1. 编写一个函数, 判断一个整数是否为素数, 如果为素数, 则返回 1, 否则返回 0, 在主函数中调用此函数找出 500~1200 之间的所有素数。

2. 编写一个函数, 求如下级数 S , 在主函数中输入 n , 并输出结果。

$$S = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \dots + \frac{1}{1+2+\dots+n}$$

3. 编写函数 `fun(n)`, n 为一个三位自然数, 判断 n 是否为水仙花数, 若是, 则返回 1, 否则返回 0。在主函数中输入一个三位自然数, 调用函数 `fun(num)`, 并输出判断结果。水仙花数是指一个 n 位数 ($n \geq 3$), 它每个位上的数字的 n 次幂之和等于它本身, 如 $1^3 + 5^3 + 3^3 = 153$ 。

4. 闰年是为了弥补因人为历法规定造成的年度天数与地球实际公转周期的时间差而设立的。补上时间差的年份(有闰日的年份)为闰年。符合以下条件之一的年份即为闰年:

(1) 能被 4 整除而不能被 100 整除;

(2) 能被 400 整除。

编写函数, 计算某个日期(某年某月某日)是本年的第几天。在主函数中输入年、月、日, 调用该函数, 并输出结果。

5. 编写 `fun()` 函数, 该函数的功能是: 根据形参 n 的值, 输出如下所示的金字塔图形。如 $n=5$, 输出图形如下。编写 `main()` 函数, 输入一个整数 n ($1 \leq n \leq 15$), 并调用 `fun()` 函数。

Please input n:5

```

1
222
3333
4444444
55555555
4444444
33333
222
1

```

6. 编写 fun() 函数, 其功能是: 判断形参 n 存放的数是否为素数。主函数将 100 以内的质对数(两质数之差为 2, 称此对数为质对数)和质对数的个数输出到屏幕上。

7. 编写 fun() 函数, 该函数的功能是: 将两个两位正整数 a、b 合并成一个新整数, 放在 c 中, 并返回给主函数。合并的方式是: 将 a 的十位和个位依次放到 c 的千位和十位, b 的十位和个位依次放到 c 的百位和个位。编写 main() 函数, 输入两个数, 并调用 fun() 函数, 输出新数。例如, 输入的第 1 个数为 45, 第 2 个数为 32, 则新数为 4352。

8. 编写递归函数, 其功能是将 1~255 之间的任意整数转换为二进制数。

9. 编写 fun() 函数, 该函数的功能是: 计算如下公式前 n 项的值, 并将结果作为返回值返回。编写 main() 函数, 输入 n(n 为大于 0 的整数), 调用 fun() 函数, 输出结果。

$$\text{sum} = \frac{1}{2^2} + \frac{3}{4^2} + \frac{5}{6^2} + \cdots + \frac{2 \times n - 1}{(2 \times n)^2}$$

10. 编写 fun() 函数, 该函数的功能是: 判断形参 x 是否是素数。编写 main() 函数, 输入一个数 n(n>=2), 调用 fun(), 输出距离 n 最近的一个素数。例如:

输入 n=11, 则输出 11(自身距离自身最近);

输入 n=9, 则输出 7(9 与前、后的素数距离相同, 则输出较小的);

输入 n=16, 则输出 17(16 之前的素数是 13, 之后的素数是 17, 其距离 17 比 13 近)。

第 8 章 自定义数据类型

存储一位学生的成绩时，可以使用整型变量；存储一位学生的姓名时，可以使用字符串；存储一个班学生的成绩时，可以使用整型数组。但在教师工资管理系统中，教师工资的信息包括：教师编号、教师姓名、教师职称、基本工资及扣款等，由于这些数据的类型各不相同，要想对这些数据进行统一管理，仅靠基本数据类型和数组很难实现。因此，C 语言提供了另外两种自定义数据类型：结构体和共用体。本章将围绕这两种数据类型进行讲解。

学习目标

- 掌握结构体的定义及其变量的定义、初始化和引用
- 掌握结构体数组的使用
- 掌握共用体的定义及其变量的定义、初始化和引用
- 理解结构体和共用体的内存分配机制

8.1 结构体

在教师工资管理系统中，一位教师的工资信息包括：教师编号、教师姓名、教师职称、基本工资及扣款等，要求能够输入、输出和存储。由于每项类型均不同，因此需要使用结构体来实现。

8.1.1 结构体的定义

结构体的基本类型不同，可以直接定义相应变量，而结构体中有哪些项，由用户自己定义，再在此基础上定义基于此结构体的变量。结构体中的项称为结构体“成员”，一个结构体就是由若干个“成员”构成的，定义结构体的关键字是 **struct**，一般形式如下：

```
struct 结构体类型名
{
    数据类型 成员名 1;
    数据类型 成员名 2;
    ...
    数据类型 成员名 n;
};
```

struct 是定义结构体的关键字，其后定义“结构体类型名”，在花括号中定义结构体类型的成员，每个成员由“数据类型”和“成员名”共同组成，最后花括号后的分号不能省略。例如，一组由教师编号 (TeaNum)、教师姓名 (TeaName)、教师职称 (TeaTitle)、基本工资 (BasePay)、课时数 (ClassHours)、每节课课时费 (ClassHoursSubsidy)、扣款 (Debit)、工资 (Salary) 组成的教师工资信息，在案例中的定义语句如下：

```
struct TeaSalary
{
    int TeaNum;           /*教师编号*/
    char TeaName[13];     /*教师姓名*/
    char TeaTitle[7];     /*教师职称*/
    int BasePay;          /*基本工资*/
    int ClassHours;       /*课时数*/
    int ClassHoursSubsidy; /*每节课课时费*/
    int Debit;            /*扣款*/
    int Salary;           /*工资*/
};
```

如同在说明和调用函数之前要先定义函数一样，由于结构体是一种“构造”而成的数据类型，所以在说明和使用之前必须先进行定义，即需要“构造”。由于结构体类型不是变量，不属于任何函数，所在其往往定义在所有函数之外。

需要注意的是，结构体类型中的成员也可以是一个结构体变量，例如，在学生信息中有出生日期一项，出生日期又可以有 year、month、day 三项，具体代码如下：

```
struct Date
{
    int year;
    int month;
    int day;
}
struct Student
{
    int num;
    char name[10];
    char sex;
    struct Date birthday;
};
```

上述代码先定义结构体类型 Date，该结构体类型由 year、month、day 三个成员组成；然后定义结构体类型 Student，其中的成员 birthday 是 Date 结构体类型的。Student 的结构如图 8-1 所示。

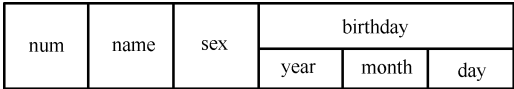


图 8-1 Student 的结构

8.1.2 结构体变量的定义与初始化

1. 结构体变量的定义

结构体类型仅相当于一个模型，其中并无具体数据，系统也不会为它分配内存空间。为了在程序中使用结构体类型的数据，应该为结构体类型定义属于这种类型的变量，并在其中存放具体的数据。定义结构体变量有三种方式。

(1) 先定义结构体类型，再定义结构体变量

“关键字”和“结构体类型名”共同构成了结构体数据类型，定义好结构体数据类型后，就可以定义结构体变量，格式如下：

```
struct 结构体类型名 变量名;
```

例如：

```
struct TeaSalary ts1,ts2;
```

语句中定义了结构体变量 `ts1` 和 `ts2`，它们可以分别存储教师工资信息的变量，分别占据一块连续的内存空间。

(2) 在定义结构体类型的同时定义结构体变量

此方式的作用与第 1 种方式相同，语法格式如下：

```
struct 结构体类型名
{
    数据类型 成员名 1;
    数据类型 成员名 2;
    ...
    数据类型 成员名 n;
}结构体变量列表;
```

例如：

```
struct Student
{
    int num;
    char name[10];
    char sex;
}stu1,stu2;
```

语句中在定义结构体类型 `Student` 的同时，也定义了结构体类型变量 `stu1` 和 `stu2`。其中变量 `stu1` 和 `stu2` 中包含的成员的数据类型相同。

(3) 直接定义结构体变量

除了上述两种方式，还可以直接定义结构体变量。定义方式是在第 2 种方式中省略“结构体类型名”，例如：

```
struct
{
    int num;
    char name[10];
    char sex;
}stu1,stu2;
```

代码中同样定义了结构体变量 `stu1` 和 `stu2`，但这种方式定义的结构体变量没有类型名，称为匿名结构体。

2. 结构体变量的初始化

由于结构体变量中存储的是一组类型不同的数据，因此，为结构体变量初始化的过程，其实就是为结构体中各个成员初始化的过程。结构体变量初始化的方式可分为以下两种。

(1) 在定义结构体类型和结构体变量的同时对结构体变量初始化，具体代码如下：

```
struct Student
{
    int num;
    char name[10];
    char sex;
}stu={20180101,"Mike",'F'};
```

此代码在定义结构体变量 `stu` 的同时，就为其中的成员初始化。

(2) 定义好结构体类型后，对结构体变量初始化，具体代码如下：

```
struct Student
{
    int num;
    char name[10];
    char sex;
};
struct Student stu={20180101,"Mike",'F'};
```

此代码先定义了一个结构体类型 `Student`，然后在定义结构体变量 `stu` 时，为其中的成员初始化。

8.1.3 结构体变量的引用

定义结构体变量后便可以引用这个变量。引用时应遵循以下规则：在引用结构体变量时，不能将一个结构体变量作为一个整体输入和输出，只能对结构体变量中的各个成员分别输入和输出。

例如，已定义 `stu1` 和 `stu2` 为结构体变量，并且它们已有值，则不能使用以下语句引用：

```
printf("%d,%s,%c",stu1);
```

也不能使用以下语句整体读入结构体变量：

```
scanf("%d,%s,%c",&stu1);
```

但可以通过赋值语句，把一个结构体变量的值赋给同类型的结构体变量，例如：

```
stu2=stu1;
```

`stu2` 和 `stu1` 具有相同的内容。

引用结构体变量中成员的方式如下：

```
结构体变量名.成员名
```

例如：

```
stu1.num        /*第 1 个人的学号*/
stu2.sex        /*第 2 个人的性别*/
```

如果成员本身又是一个结构体，则必须逐级找到最低级的成员才能使用。只能对最低级的成员进行赋值、存取、运算等，例如：

```
stu1.birthday.month
```

可以对变量的成员赋值，例如：

```
stu1.num=20180101
```

8.2 结构体数组

一个 TeaSalary 结构体变量可以存储一位教师的工资信息，但学校不止一位教师的工资信息需要存储，因此需要使用数组。此时的数组称为结构体数组，数组的每个元素是 TeaSalary 结构体变量。与前面章节的数组不同，结构体数组中的每个元素都是结构体类型，它们都有若干个成员项。

208

8.2.1 结构体数组的定义与初始化

结构体数组定义的方法和结构体相似，只需说明它为数组类型即可。例如：

```
struct Student
{
    int num;
    char name[10];
    char sex;
}stu [5];
```

此代码定义了一个结构体数组 stu，共有 5 个元素(stu[0]~stu[4])。每个数组元素都具有 struct Student 的结构体类型。教师工资管理系统中的结构体变量的定义就是采用此种形式。当然也可以采用先定义结构体类型，后定义结构体数组，或直接定义结构体数组的方法。

结构体数组与数组类似，都是通过为元素赋值的方式完成初始化的。由于结构体数组中的每个元素都是一个结构体，因此，在为每个元素赋值时，需要将每个成员的值依次放到一对花括号中，例如：

```
struct Student
{
    int num;
    char name[10];
    char sex;
}stu[3]={
    {20180101,"Mike",'F'},
    {20180102,"Mary",'M'},
    {20180103,"Tom",'F'}};
```

当全部元素进行初始化赋值时，也可以不给出数组长度。

8.2.2 结构体数组元素的引用

结构体数组的引用是指对结构体数组元素的引用, 由于每个结构体数组元素都是一个结构体变量, 因此, 结构体数组元素的引用方式与结构体变量类似, 只是在结构体数组名后需要带上元素下标, 形式如下:

结构体数组元素名[下标].成员名;

例如, 要输出上段代码中第1个元素的name成员, 可以采用以下方式:

```
printf("%s", stu[0].name);
```

【案例 8-1】 编写代码, 完成教师工资管理系统中教师工资信息的输入功能。根据教师工资信息定义结构体类型, 再定义结构体数组, 并输入数组的每个元素。

```
#include<stdio.h>
struct TeaSalary
{
    int TeaNum;           /*教师编号*/
    char TeaName[13];     /*教师姓名*/
    char TeaTitle[7];     /*教师职称*/
    int BasePay;          /*基本工资*/
    int ClassHours;       /*课时数*/
    int ClassHoursSubsidy; /*每节课课时费*/
    int Debit;            /*扣款*/
    int Salary;           /*工资*/
} tea[3];
int main()                /*添加教师记录*/
{
    int i;
    printf("开始添加教师工资信息...\n");
    for(i=0; i<3; i++) /*输入的教师元素下标应该从当前记录数开始, 所以 i=cnt*/
    {
        printf("教师编号: ");
        scanf("%d", &tea[i].TeaNum);
        getchar();
        printf("教师姓名: ");
        gets(tea[i].TeaName);
        printf("教师职称: ");
        gets(tea[i].TeaTitle);
        printf("教师课时: ");
        scanf("%d", &tea[i].ClassHours);
        printf("教师扣款: ");
        scanf("%d", &tea[i].Debit);
        printf("_____ \n");
    }
    printf("  编号  姓名  职称  基本工资  课时数  课时费  扣款  工资\n");
    for(i=0; i<3; i++)
```

```
        {  
            printf("%-8d%-8s%-8s%-10d%-8d%-8d%-8d\n", tea[i].TeaNum,  
                tea[i].TeaName, tea[i].TeaTitle, tea[i].BasePay,  
                tea[i].ClassHours, tea[i].ClassHoursSubsidy,  
                tea[i].Debit, tea[i].Salary);  
        }  
    return 0;  
}
```

运行结果如图 8-1 所示。由于此段代码实现了将输入的教师工资信息输出的功能，但还没有实现根据教师职称计算基本工资、课时费和工资的功能，所以相应值都为 0。

```
开始添加教师工资信息....  
教师编号: 1001  
教师姓名: 张华  
教师职称: 讲师  
教师课时: 60  
教师扣款: 314  
  
教师编号: 1002  
教师姓名: 李琼  
教师职称: 教授  
教师课时: 54  
教师扣款: 286  
  
教师编号: 1004  
教师姓名: 王芳  
教师职称: 副教授  
教师课时: 50  
教师扣款: 360
```

编号	姓名	职称	基本工资	课时数	课时费	扣款	工资
1001	张华	讲师	0	60	0	314	0
1002	李琼	教授	0	54	0	286	0
1004	王芳	副教授	0	50	0	360	0

图 8-1 运行结果

210

8.3 共用体

“共用体”与“结构体”有一些相似之处，但两者也有着本质的区别。在“结构体”中各成员有各自的内存空间，一个结构体变量的总长度是各成员长度之和。而在“共用体”中，各成员共享一段内存空间，一个共用体变量的长度等于成员中最大成员的长度。

需要说明的是，这里所谓的“共享”不是指把多个成员同时装入一个共用体变量内，而是指该共用体变量可被赋予任意一个成员值，但每次只能赋予一种值。赋予新值，则会删除旧值。

一个共用体类型必须在定义后才能将变量说明为该共用体类型。当一个共用体被说明时，编译程序自动产生一个变量，其长度为共用体中最大变量的长度。

共用体访问成员的方法与结构体相同。

8.3.1 共用体的定义

在 C 语言中，共用体与结构体一样，都属于自定义类型。共用体的定义与结构体十分相似，形式如下：

```
union 共用体类型名
{
    数据类型 成员名 1;
    数据类型 成员名 2;
    ...
    数据类型 成员名 n;
};
```

union 是定义共用体的关键字，其后是“共用体类型名”，花括号中定义共用体的成员，每个成员由“数据类型”和“成员名”共同组成，例如：

```
union data
{
    char c;
    int i;
    float x;
};
```

此代码定义了一个名为 **data** 的共用体，该共用体由三个不同类型的成员组成，分别为 **c**、**i** 和 **x**。它们占用同一个起始地址的存储空间，占用内存等于最大的成员的长度，内存占用情况如图 8-2 所示。



图 8-2 data 共用体内存占用情况

需要注意的是，共用体定义后即可进行共用体变量说明，被说明为 **data** 类型的变量可以存放字符型变量 **c**、整型变量 **i** 或浮点型变量 **x**，只可选择三者之一，但不能把三者同时赋予它。

另外，共用体还可以出现在结构体中，它的成员也可以是结构体，例如：

```
struct
{
    int age;
    char addr[50];
    union{
        int i;
        char ch[10];
    }x;
}s[10];
```

此代码中共用体作为结构体的一个成员，若要访问结构体变量 **s[0]** 中共用体 **x** 的成员 **i**，可以写成：

```
s[0].x.i;
```

8.3.2 共用体变量的定义与初始化

共用体变量的定义与结构体变量的定义相似，如果要定义两个 `data` 类型的共用体变量 `a` 和 `b`，则可以使用以下三种方式。

(1) 先定义共用体类型，再定义共用体变量，例如：

```
union data
{
    char c;
    int i;
    float x;
};
union data a,b;
```

(2) 定义共用体类型的同时定义共用体变量，例如：

```
union data
{
    char c;
    int i;
    float x;
}a,b;
```

(3) 直接定义共用体变量，例如：

```
union
{
    char c;
    int i;
    float x;
}a,b;
```

经过定义后，`a` 和 `b` 变量的长度应等于 `data` 的成员中最大成员的长度。共用体变量的初始化与结构体变量的初始化相似，读者可对照使用。

8.3.3 共用体变量的引用

对共用体变量的赋值、使用都是对其变量的成员进行的。

共用体变量的成员表示如下：

共用体变量名.成员名

例如，`a` 被定义为 `data` 类型的变量后，可使用 `a.c`、`a.i` 或 `a.x`。不允许只用共用体变量名进行赋值或其他操作，也不允许对共用体变量做初始化赋值，一个共用体变量的值就是共用体变量的某一个成员的值，例如：

```
#include<stdio.h>
union data
{
```

```

char c;
int i;
float x;
}a,b;
int main()
{
    a.i=97;
    printf("%c\n",a.c);
    return 0;
}

```

此代码中，由于 data 共用体的变量 a 中的各成员共用内存，所以变量 a 的 c 成员虽然没有赋值，但在输出前有如下语句：

```
a.i=97;
```

因此，输出 a.c 的结果是'a'。

【案例 8-2】 编写程序，设计一个师生信息统计表，将教师与学生的信息统计在一个表格中，如果是学生就记录其姓名、性别、角色、所在班级，如果是教师就记录其姓名、性别、角色、所在办公室。

213

```

#include<stdio.h>
#define N 3
struct
{
    char name[15];
    int age;
    char status;
    union
    {
        int grade;
        char office[20];
    }depa;
}person[3];
int main()
{
    int i;
    for(i=0;i<N;i++)
    {
        printf("input name:\n");           /*提示语*/
        gets(person[i].name);             /*gets()函数接收带空格的姓名*/
        printf("input age:\n");
        scanf("%d",&person[i].age);
        getchar();                         /*接收上一句输入的回车符*/
        printf("input status(s or t) :\n");
        person[i].status=getchar();
        if( person[i].status=='s')         /*如果角色是学生*/
        {   getchar();                     /*接收上一句输入的回车符*/

```

```

        printf("input grade:\n");
        scanf("%d", &person[i].depa.grade);    /*共用体中的班级*/
        getchar();
    }
    else                                        /*如果角色是教师*/
    {   getchar();
        printf("input office:\n");
        gets(person[i].depa.office);          /*共用体中的办公室*/
    }
}
printf("name\t\tage status grade/office\n");
for(i=0;i<N;i++)
{
    if(person[i].status=='s')
        printf("%-15s\t%3d%3c%20d\n",person[i].name,person[i].age,
                person[i].status,person[i].depa.grade);
                                                /*对齐输出数据*/
    else
        printf("%-15s\t%3d%3c%20s\n",person[i].name,person[i].age,
                person[i].status,person[i].depa.office);
}
return 0;
}

```

```

input name:Mary
input age:34
input status(s or t) :t
input office:software
input name:Tom
input age:12
input status(s or t) :s
input grade:601
input name:Jack
input age:11
input status(s or t) :s
input grade:502
name          age status grade/office
Mary          34  t          software
Tom           12  s           601
Jack          11  s           502
Press any key to continue

```

图 8-3 运行结果

程序的运行结果如图 8-3 所示。

程序用一个结构体数组 `person` 来存放人员数据，该结构共有 4 个成员。其中，成员 `depa` 是一个共用体，这个共用体又由两个成员组成，一个为整型变量 `grade`，另一个为字符数组 `office`。在程序的第 1 个 `for` 语句中，输入人员的各项数据。先输入结构的前 3 个成员 `name`、`age` 和 `status`，然后判断 `status` 成员，如果为 `s`，则对共用体 `depa.grade` 输入（对学生赋班级名），否则对 `depa.office` 输入（对教师赋办公室名）。程序中的第 2 个 `for` 语句用于输出各成员的值。

8.4 本章小结

本章主要讲解结构体和共用体自定义构造类型。结构体允许将若干个相关的、数据类型不同的数据作为一个整体处理，并为每个数据分配不同的内存空间；而共用体中所有的成员共享同一段内存空间。通过本章的学习，读者应熟练掌握结构体和共用体的基本概念和使用方法，并将其灵活运用到实际编程中。

8.5 习题

一、单选题

1. 已知学生记录描述如下, 设变量 s 的生日是 1984 年 11 月 11 日, 则下列生日的赋值方式正确的是()。

```
struct student
{
    int no;
    char name[20];
    char sex;
    struct
    {
        int year;
        int month;
        int day;
    }birth;
};
struct student s;
```

- | | |
|---|---|
| <p>A. year=1984;
month=11;
day=11</p> <p>C. s.year=1984;
s.month=11;
s.day=11</p> | <p>B. birth.year=1984;
birth.month=11;
birth.day=11</p> <p>D. s.birth.year=1984;
s.birth.month=11;
s.birth.day=11</p> |
|---|---|
2. 定义一个结构体变量时, 系统分配给它的内存是()。
- | | |
|---|--|
| <p>A. 各成员所需内存的总和</p> <p>C. 成员中占内存最大者所需的容量</p> | <p>B. 第 1 个成员所需的内存</p> <p>D. 结构体中最后一个成员所需内存量</p> |
|---|--|
3. 以下结构体变量的定义不正确的是()。
- | | |
|---|---|
| <p>A. #define STUDENT struct student
STUDENT
{
int num;
float age;
}std1;</p> <p>C. struct
{
int num;
float age;
}std1;</p> | <p>B. struct student
{
int num;
float age;
}std1;</p> <p>D. struct
{
int num;
float age;
}student;
struct student std1;</p> |
|---|---|

4. 阅读以下程序,叙述不正确的是()。

```
struct stu
{
    int a;
    float b;
}stutype
```

- A. struct 是结构体的关键字 B. struct stu 是用户定义的结构体类型名
C. stutype 是用户定义的结构体类型名 D. a 和 b 都是结构体成员名
5. C 语言结构体变量在程序执行期间()。
- A. 所有成员一直驻留在内存中 B. 只有一个成员驻留在内存中
C. 部分成员驻留在内存中 D. 没有成员驻留在内存中
6. 有如下定义(假设一个整型变量占 4 个字节),则结构体变量 b 占用内存的字节数是()。

```
struct data
{
    int i;
    char ch;
    double f;
}b;
```

- A. 1 B. 2 C. 11 D. 13

7. 以下程序的运行结果是()。

```
#include "stdio.h"
main()
{
    struct data
    {
        int year,month,day;
    }today;
    printf("%d\n",sizeof(struct date));
}
```

- A. 6 B. 8 C. 10 D. 12

8. 根据以下定义,能打印出字母 M 的语句是()。

```
struct person{ char name[9];
               int age;
};
struct person class[10]={"John",17
                          "Paul",19
                          "Mary",18
                          "adam",16
};
```

- A. printf("%c\n",class[3].name); B. printf("%c\n",class[3].name[1]);
C. printf("%c\n",class[2].name[1]); D. printf("%c\n",class[2].name[0]);

9. 以下程序的运行结果是()。

```
main()
{
    struct cmplx { int x;
                  int y;
                  } cnum[2]={1,3,2,7};
    printf("%d\n",cnum[0].y/cnum[0].x*cnum[1].x);
}
```

- A. 0 B. 1 C. 3 D. 6

10. 以下程序的运行结果是()。

```
struct KeyWord
{
    char Key[20];
    int ID;
}kw[]={ "void",1, "char",2, "int",3, "float",4, "double",5};
main()
{
    printf("%c,%d\n",kw[3].Key[0],kw[3].ID);
}
```

- A. i,3 B. n,3 C. f,4 D. l,4

11. 根据下面的定义,能输出 Mary 的语句是()。

```
struct person
{
    char name[9];
    int age;
};
struct person class[5]={"John",17,"Paul",19,"Mary",18,"Adam",16};
```

- A. printf("%s\n",class[1].name); B. printf("%s\n",class[2].name);
C. printf("%s\n",class[3].name); D. printf("%s\n",class[0].name);

12. 定义以下结构体数组,“printf("%s,%d\n",x[0].name,x[1].birthday.year);”的输出结果是()。

```
struct date
{
    int year;
    int month;
    int day;
};
struct s
{
    struct date birthday;
    char name[20];
}x[4]={{2008,10,1,"guangzhou"},{2009,12,25,"Tianjin"}};
```

A. guangzhou,2009

B. guangzhou,2008

C. Tianjin,2008

D. Tianjin,2009

13. 有如下共用体定义, 则不正确的语句是()。

```
union date {char ch; int x;}a;
```

A. a.x=13; a.ch='c';

B. a=13,'c';

C. a.x=13;

D. a.x='c';

14. 以下程序的输出结果是()。

```
#include <stdio.h>
struct STU
{
    char num[10];
    float score[3];
}s[3]={{"20021",90,95,85},{ "20022",95,80,75},
      {"20023",100,95,90}};
void main()
{
    int i; float sum=0;
    for(i=0;i<3;i++)
        sum=sum+s[i].score[i];
    printf("%6.2f\n",sum);
}
```

A. 270.00

B. 250.00

C. 285.00

D. 260

15. 以下程序的输出结果是()。

```
#include <stdio.h>
struct abc
{int a;int b;int c;};
main()
{ struct abc s[2]={1,2,3},{4,5,6}};
  int t;
  t=s[0].a+s[1].b;
  printf("%d\n",t);
}
```

A. 5

B. 6

C. 7

D. 8

16. 以下程序的输出结果是()。

```
#include <stdio.h>
union myun
{ struct
  {int x;int y;int z;}u;
  int k;
}a;
main()
{ a.u.x=4;
  a.u.y=5;
  a.u.z=6;
  a.k=0;
```

```
printf("%d\n",a.u.x);
}
```

- A. 5 B. 6 C. 4 D. 0

二、填空题

1. 以下程序的运行结果是_____。

```
#include<stdio.h>
struct n{
    int x;
    char c;
};
void func(struct n b)
{
    b.x=20;
    b.c='y';
}
main()
{
    struct n a={10,'x'};
    func (a);
    printf("%d,%c",a.x,a.c);
}
```

2. 以下程序的运行结果是_____。

```
#include<stdio.h>
main()
{
    struct EXAMPLE {
        struct {
            int x;
            int y;
        }in;
        int a;
        int b;
    }e;
    e.a=1; e.b=2;
    e.in.x=e.a*e.b;
    e.in.y=e.a+e.b;
    printf("%d,%d\n",e.in.x,e.in.y);
}
```

3. 以下程序用来输出结构体变量 bt 所占内存单元的字节数，请在程序的对应位置填空。

```
struct ps
{
    double i;
    char arr[20];
};
main()
{
```

```

    struct ps bt;
    printf("bt size:%d\n", _____);
}

```

4. 以下程序用来按姓名查询排名和平均成绩。查询可连续进行，直到输入 0 时结束，请在程序的对应位置填空。

```

#include <stdio.h>
#include <string.h>
#define NUM 4
struct student
{
    int rank;
    har *name;
    float score;
};
_____stu[]={3,"Tom",89.3,4,"Mary",78.2,1,"Jack",95.1,2,"Jim",90.6};
void main()
{
    char str[10];
    int i;
    do{ printf("Enter a name:");
        scanf("%s",str);
        for(i=0;i<NUM;i++)
            if(_____)
            {
                printf("name:%8s\n",stu[i].name);
                printf("rank:%3s\n",stu[i].rank);
                printf("average:%5.1f\n",stu[i].score);
                _____;
            }
        if(i>=NUM) printf("Not found\n");
    }while(strcmp(str,"0")!=0);
}

```

5. 设有三人的姓名和年龄存储在结构体数组中，以下程序输出三人中年龄居中者的姓名和年龄，请在程序的对应位置填空。

```

static struct man
{
    char name[20];
    int age;
}person[ ]={"li-ming",18,"wang-hua",19,"zhang-ping",20};
void main()
{
    int i,j,max,min;
    max=min=person[0].age;
    for(i=1;i<3;i++)
        if(person[i].age>max) _____;
        else if(person[i].age<min) _____;
    for(i=0;i<3;i++)
        if(person[i].age!=max_____person[i].age!=min)

```

```

        {
            printf("%s %d\n", person[i].name, person[i].age);
            break;
        }
    }
}

```

6. 以下程序使用比较计数法对结构体数组 **a** 按字段 **num** 进行降序排列。比较计数法的基本思想是:通过另一个字段 **con** 记录 **a** 中小于某一特定关键字的元素个数。待算法结束, **a[i].con** 就是 **a[i].num** 在 **a** 中的排列位置。请在程序的对应位置填空。

```

#include<stdio.h>
#define N 8
struct c
{
    int num;
    int con;
}a[16];
main()
{
    int i, j;
    for(i=0;i<N;i++)
    {
        scanf("%d",&a[i].num);
        _____;
    }
    for(i=N-1;i>=1;i--)
        for(j=i-1;j>=0;j--)
            if(a[i].num<a[j].num)
                _____;
            else
                _____;
    for(i=0;i<N;i++)
        printf("%d,%d\n",a[i].num,a[i].con);
}

```

三、上机操作题

1. 编写程序, 定义一个结构体变量(包括年、月、日), 计算该日是本年中的第几天, 注意闰年问题。
2. 编写程序, 设计一个函数 **days()** 实现第 1 题的计算。由主函数将年、月、日传递给 **days()** 函数, 计算后将天数传回主函数, 并输出。
3. 编写程序, 设计一个函数 **print()**, 输出 5 位学生的数据记录, 每个记录包括 **num**、**name**、**score[3]**, 用主函数输入这些记录, 用 **print()** 函数输出这些记录。
4. 在第 3 题的基础上, 编写一个函数 **input()**, 用来输入 5 位学生的数据记录。
5. 有 10 位学生, 每位学生的数据包括学号、姓名、3 门课的成绩。从键盘输入 10 位学生的数据, 要求打印出 3 门课的平均分, 以及平均成绩最高的学生的数据(包括学号、姓名、3 门课的成绩、3 门课的平均分)。

第9章 指 针

为了便于计算机系统对内存的管理，内存中的每个字节都有一个唯一的编号，称为内存地址。程序在编译运行时会为每个变量分配内存空间，即每个变量在内存中的地址是唯一的，可将变量在内存中的地址保存在另一个特殊的变量(指针变量)中。正确地使用指针，可以使程序代码更为简洁紧凑、高效灵活。指针是 C 语言的精髓，同时也是 C 语言中最难掌握的内容之一。本章主要讲解指针与指针变量、指针与数组、指针与函数的关系等。

学习目标

- 掌握指针与指针变量的概念
- 掌握指针的运算方法
- 掌握如何使用指针引用数组中的数据
- 了解指针与函数的关系
- 掌握指针作为参数的作用
- 了解二级指针的概念
- 掌握申请内存与释放内存的方法
- 了解链表的概念
- 理解链表的创建、删除、插入等基本操作

9.1 指针的概念

9.1.1 指针与指针变量

1. 指针

【案例 9-1】 编写程序，输出整型变量 a 和 b 在内存中的地址。

```
#include<stdio.h>
int main()
{
    int a=100,b=200;
    printf("%x,%x\n",&a,&b);
    return 0;
}
```

```
18ff44,18ff40
Press any key to continue
```

图 9-1 运行结果

程序的运行结果如图 9-1 所示。

在程序中，没有输出变量 a 和 b 的值，而是通过符号“&”来获取变量的地址。从结果可以看出，编译器为变量

a 分配了 4 个字节连续地址的存储空间(18ff44H~18ff41H)，18ff44H 就是这个变量在内存中的地址。因为通过变量的地址可以找到该变量所在的存储空间，所以说该变量的地址指向该变量所在的存储空间，该地址是指向该变量的指针。

若将存储空间看成公寓，那么存储单元好比是公寓中的床位，地址好比是公寓中的床位号，而存储空间中存储的数据就相当于住进公寓的学生。每位学生对应一个床位号，通过床位号可以找到唯一的学生。

2. 指针变量的定义

指针指示某个变量所在的存储空间，即内存地址，相应地，可以用指针变量存储这个指针(内存地址)。指针变量遵循基本类型变量的原则，即“先定义，后使用”，其定义的语法形式如下：

```
变量类型 *变量名;
```

其中，“变量类型”定义指针指向数据的类型，即指针变量存储的地址是哪种数据类型的地址，变量名前的“*”表示该变量是一个指针变量。例如：

```
int *p
```

“*”表示 p 是一个指针变量，int 表示该指针变量指向一个 int 型数据所在的地址，即指针变量 p 中存储的地址只能是整型变量的地址。

在定义指针变量时需要注意如下内容。

指针变量前面的“*”表示该变量的类型为指针型变量。例如：

```
float *p;
```

指针变量名是“p”，而不是“*p”。

在定义指针变量时必须指定基类型。需要特别注意的是：只有整型变量的地址才能放到指向整型变量的指针变量中。下面的赋值语句是错误的：

```
float a;  
int *pointer_1;  
pointer_1=&a;
```

3. 指针变量的初始化

指针变量必须要指向某个变量，即必须存储某个内存地址才有意义，指针变量的赋值有如下两种方法。第 1 种是接收变量的地址为其赋值，例如：

```
int a=100;  
int *p;  
p=&a;
```

第 2 种是将其他指针的值赋给指针变量，这样两个指针变量就可指向同一块存储空间，例如：

```
int *p,*q;  
p=q;
```

在第 1 种方法中出现的“&”是取地址运算，作用是获取变量 a 的地址。

第 1 种方法也可以在定义的同时为指针变量赋值，例如：

```
int a=100;
int *p=&a;
```

在对指针变量赋值时需要注意：

指针变量中只能存放地址(指针)，不要将一个整数赋给一个指针变量。下面的赋值语句是错误的：

```
*p=100;          /*p 是指针变量，100 是整数，不合法*/
```

赋给指针变量的变量地址不能是任意类型的，只能是与指针变量的基类型具有相同类型的变量的地址。

9.1.2 指针变量的引用

指针变量的引用就是根据指针变量中存放的地址访问该地址对应的变量。访问指针变量中指针所指变量的方式非常简单，只需要在指针变量名前添加一个取值运算符“*”即可，其语法格式如下：

*指针变量名

例如：

```
int a=100;
int *p=&a;
printf("%d\n", *p); /*输出指针变量指向的地址中存储的数据*/
```

第 3 条语句的作用与“printf("%d\n",a);”的作用相同，只是第 3 条语句是间接寻址访问，而“printf("%d\n",a);”是直接寻址访问。

虽然间接访问比直接访问复杂，但在有些场合只能使用间接访问，例如：

在用户动态申请一块内存空间时，因为该内存空间没有对应的变量名，所以只能通过首地址对其进行操作(关于内存的申请与回收将在后面讲解)；

在通过被调用函数改变主调函数变量的值时，由于值只能由实参向形参单向传递，所以被调函数无法通过改变形参的值去改变主调函数中变量的值，只能通过间接访问指针指向的内存空间来改变主调函数中变量的值。

【案例 9-2】 编写程序，实现通过指针变量访问整型变量。

```
#include <stdio.h>
void main()
{
    int a,b;
    int *pointer_1,*pointer_2;
    a=100;b=10;
    pointer_1=&a; /*把变量 a 的地址赋给 pointer_1 */
    pointer_2=&b; /*把变量 b 的地址赋给 pointer_2 */
    printf("%d,%d\n",a,b);
    printf("%d,%d\n",*pointer_1,*pointer_2);
}
```

程序的运行结果如图 9-2 所示。程序中将变量 a 和 b 的地址分别赋给指针变量 pointer_1 和 pointer_2，此时，指针变量存储的内容是变量 a 和 b 在内存中的地址，在内存中的存储形式如图 9-3 所示。

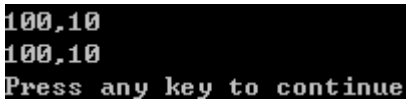


图 9-2 运行结果

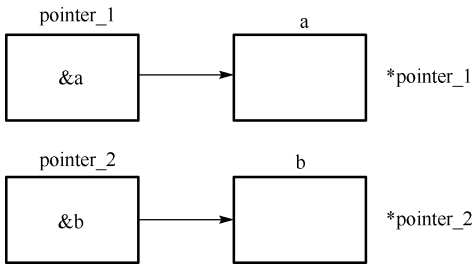


图 9-3 在内存中的存储形式

9.2 指针的运算

指针是一种数据类型，指针变量是能存储值的变量，那么它与其他变量一样，也可以进行运算。指针变量的运算主要包括：取地址运算、取内容运算、赋值运算、算术运算和关系运算。

1. 取地址运算

取地址运算符“&”是单目运算符，其功能是取变量的地址，例如：

```
i_p=&i; /*变量 i 的地址送给指针变量 i_p*/
```

2. 取内容运算

取内容运算符“*”是单目运算符，用来表示指针变量所指单元的内容，在“*”运算符之后必须跟指针变量，例如：

```
j=*i_p; /*将 i_p 所指的单元的内容赋给变量 j*/
```

注意“*”在此处是取内容运算符，需要与算术运算的乘法运算符号“*”区别开来。区别的方法是看“*”的操作数的个数，如果有左和右操作数，则“*”是乘号，如果只有右操作数，则“*”是取内容运算符，例如：

```
j=i*j; /*有左和右操作数，是乘号*/
j=*p; /*只有右操作数，是取内容运算符*/
```

3. 赋值运算

把一个指针变量的值赋给指向相同类型变量的另一个指针变量，这样两个指针变量就指向同一块存储空间，例如：

```
int x,*p_x,*p_y;
p_x=&x;
p_y=p_x;
```

指针 `p_x` 的值为变量 `x` 的地址。赋值语句将指针 `p_x` 的值赋给指针 `p_y`，现在指针 `p_x` 和指针 `p_y` 指向同一个变量 `x`。

把数组的首地址赋给指针变量，例如：

```
int a[5],*pa;
pa=a;          /*数组名代表数组的首地址，所以此处如果改为 pa=&a;是错误的*/
```

由于数组元素占用内存中一块连续的存储单元，数组名就表示数组的首地址，所以可以将数组名直接赋给一个指向数组的指针变量 `pa`。注意，在赋值语句的数组名 `a` 前面不用取地址运算符 “&”。

4. 算术运算

数值变量可以进行加、减、乘、除算术运算。而对于指针变量，由于它保存的是一个内存地址，所以对两个指针进行乘、除运算是没有意义的。指针的算术运算主要指指针的移动，即通过指针递增或递减，加上或减去某个整数值来移动指针指向的内存位置，这一点对数组元素下标的移动十分有用。

使用递增或递减运算符（++或--）将指针递增或递减，例如：

```
int *ptrnum,arr_num[10];
ptrnum=arr_num;
ptrnum++;
```

其中，指针 `ptrnum` 指向整型数组 `arr_num`，即存储数组中第 1 个元素的地址。然后，使用 “++” 运算符递增该指针。这意味着，`ptrnum` 此时指向 `arr_num[0]` 地址的后一个连续地址，即数组中下一个元素的地址。应注意，数组指针变量向前或者向后移动 1 个位置和地址加 1 或减 1 在概念上是不同的。指针变量加 1，即向后移动 1 个位置，表示指针变量指向下一个元素的首地址，而不是在原地址的基础上加 1。所以，一个类型为 `T` 的指针的移动，以 `sizeof(T)` 为移动单位，如果 `ptrnum` 的地址是 `18fe40H`，那么 `ptrnum++` 后的地址是 `18fe44H`。因为 VC++ 6.0 中，整型变量占用 4 个字节。

当指针加上或者减去某个整数值时，指针向后或者向前移动对应个数的数据单元，例如：

```
ptrnum=&arr_num[5];
ptrnum=ptrnum-2;
```

此处指针首先指向数组的第 6 个元素，然后将指针减 2。这意味着 `ptrnum` 此时指向数组的第 4 个元素，即 `arr_num[3]`。

两个指针也可以相减，如 `p1-p2`，但只有 `p1` 和 `p2` 都指向同一个数组中的元素时才有意义。

当表达式中既有取内容运算，又有++或--运算时，由于两种运算都是单目运算，优先级相同，在运算时要注意结合性都是自右向左，例如：

```
int a[]={1,3,5,7};
int *p=a;
printf("%d\n",++*p);  /*先取内容，再将内容自增 1，结果为 2*/
```

如果把最后一句改为：

```
printf("%d\n",**++p); /*p 所指向的地址先自增 1，指向第 2 个元素，再取内容，结果为 3*/
```

则两句的结果是完全不一样的。

5. 关系运算

两个指针在有意义的情况下，可以进行比较运算。如可以比较两个指针，看它们是否相等，即这两个指针是否指向同一个变量，例如：

```
#include <stdio.h>
int main()
{
    int *ptrnum1,*ptrnum2;
    int value=1;
    ptrnum1=&value;
    value+=10;
    ptrnum2=&value;
    if (ptrnum1==ptrnum2)
        printf ("\n 两个指针指向同一个地址\n");
    else
        printf("\n 两个指针指向不同的地址\n");
    return 0;
}
```

其中，声明两个指针变量 `ptrnum1` 和 `ptrnum2`，另外还声明了一个 `int` 类型的变量 `value`，初始值为 1，接着将变量 `value` 的地址赋给指针 `ptrnum1`，然后将 `value` 加 10，再将 `value` 的地址赋给指针 `ptrnum2`。通过 `if` 语句判断 `ptrnum1` 和 `ptrnum2` 是否相等，即判断它们是否指向同一个地址。因为指针 `ptrnum1` 和 `ptrnum2` 存储的都是变量 `value` 的地址，所以即使变量 `value` 的值增加了 10，地址也仍保持不变。因此，相等条件的值为真，输出结果为“两个指针指向同一个地址”。

当然，两个指向相同类型变量的指针也可以进行大小比较，但通常在实际应用中无太大的意义。

9.3 指针与数组

9.3.1 指向一维数组的指针

在几乎所有使用数组名的表达式中，数组名的值是一个指针常量，也就是数组中第 1 个元素的地址。因此，一维数组的数组名表示的是该数组中第 1 个元素的存储地址，数组名的类型是“指向数组元素存储类型的常量指针”。数组名是一个指针常量，而不是指针变量，因此，数组名的值是不能修改的，但可以将指针常量的值赋给一个相同类型的指针变量，此时指针变量就可以指向该数组。将指针变量指向一维数组的一般格式如下：

```
变量类型 *变量名=数组名；
```

例如：

```
int a[5]={1,2,3,4,5};
int *p=a;
```

实际上，定义语句“`int *p=a`”与赋值语句“`int *p=&a[0]`”是等价的，都是将数组中第 1 个元素的地址赋给指针变量。赋值后 `p` 与数组名等价，访问数组元素除使用下标外，还可以使用指针变量，例如：

```
printf("%d",*(p+i));
```

与语句

```
printf("%d",a[i]);
```

是相同的，都是输出数组第 `i` 个元素的值。指针与数组的关系如图 9-4 所示。

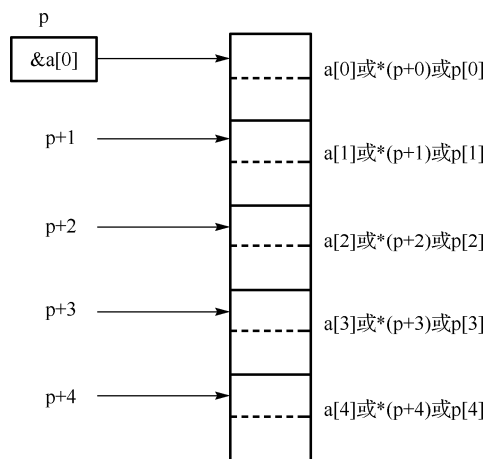


图 9-4 指针与数组的关系

若 `p` 指向数组 `a`，要使用指针获取数组元素 `a[2]` 的值，可以使用以下两种方式。

① 移动指针方式，使指针指向 `a[2]`，获取指针指向元素的值，例如：

```
p=p+2;
printf("%d",*p);
```

② 不改变指针指向，通过数组元素指针间的关系，运算指针，并取值，例如：

```
printf("%d",*(p+2));
```

关于指针变量指向一维数组后的相关操作可参照 9.2 节指针的运算。但需要注意，两个指针相加是没有意义的。

【案例 9-3】 一维数组 `array` 中包含 10 个元素，编写程序，求数组中所有元素的和，以及其中的最大值和最小值，用指针实现对数组元素的访问。

```
#include <stdio.h>
int main()
{
    int array[10];
    int *p=array,max,min,i,sum=0; /*定义变量，将数组首地址赋值给指针变量*/
    for(i=0;i<10;i++)
```

```

        scanf("%d",p+i);
max=min=*p;                                /*将第1个元素的值赋给最大值和最小值*/
for(i=0;i<10;i++)
{
    if(max<*p)
        max=*p;
    if(min>*p)
        min=*p;
    sum+=*p++;                            /*将每个元素累加到 sum 中*/
}
printf("sum=%d,max=%d,min=%d\n",sum,max,min);
return 0;
}

```

程序的运行结果如图 9-5 所示。

程序中指针变量 `p` 指向数组首地址后,输入元素使用首地址加下标的形式,通过变量 `i` 的变化访问每个元素。在求和时使用指针变量自增运算,也可以通过指针变量每次循环自增 1 的方式实现所有元素的访问。

```

2 3 4 5 1 10 9 8 7 6
sum=55,max=10,min=1
Press any key to continue

```

图 9-5 运行结果

【案例 9-4】 使用指针实现冒泡排序法,将具有 10 个元素的一维整型数组 `array[10]` 中的元素按由小到大的顺序排列(`array[10]={98,100,45,-54,0,65,106,84,20,38}`)。

案例分析:通常在使用指针实现一维数组的某些操作的程序中,需要定义一个数组元素类型的指针变量,初始化使该指针变量指向数组的第 1 个元素。通过指针变量的变化来访问所指向的存储空间中的数据,然后进行相应操作以满足要求。

```

#include <stdio.h>
#define N 10
int main()
{
    int array[N]={98,100,45,-54,0,65,106,84,20,38};
    int *p=array,t,i;
    printf("Sort befor:\n");
    for(;p<array+N;p++)
        printf("%5d",*p);

    for(i=0;i<N-1;i++)
    {
        for(p=array;p<array+N-i-1;p++)
        {
            if(*p>*(p+1))
            {
                t=*p;
                *p=*(p+1);
                *(p+1)=t;
            }
        }
    }
}

```

```

    }
    printf("\nSort after:\n");
    for(p=array;p<array+N;p++)
        printf("%5d",*p);
    printf("\n");
    return 0;
}

```

程序的运行结果如图 9-6 所示。

```

Sort befor:
 98 100 45 -54 0 65 106 84 20 38
Sort after:
-54 0 20 38 45 65 84 98 100 106
Press any key to continue

```

图 9-6 运行结果

9.3.2 指向二维数组的指针

230

在之前的案例中，我们学习了如何用指针引用一维数组。二维数组与多维数组同样有地址，也可以使用指针进行引用。只是因为其逻辑结构较一维数组更加复杂，所以操作也相对复杂。在实际编程中使用较多的是一维数组与二维数组，下面主要介绍指针与二维数组的关系。

要定义一个 2 行 3 列的二维数组，例如：

```
int a[2][3]={1,2,3},{4,5,6}};
```

其中，**a** 是二维数组的数组名，该数组中包含 2 行数据，分别为{1,2,3}和{4,5,6}。从数组 **a**[][] 的形式可以看出，这 2 行数据分别为一个一维数组，所以二维数组又可视作数组元素为一维数组的一维数组，即数组的数组。数组 **a**[][] 的逻辑示意如图 9-7 所示。

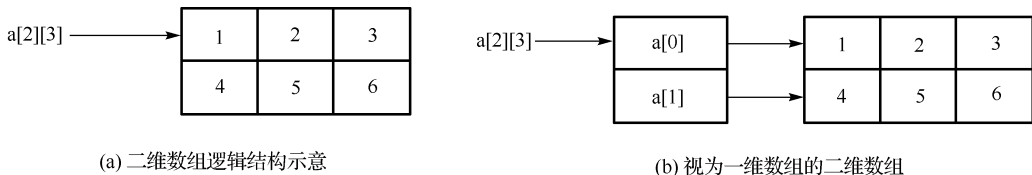


图 9-7 数组 **a**[][] 的逻辑示意

根据图 9-7(b)可以看出，与一维数组一样，二维数组的数组指针同样指向数组中的第 1 个元素，只是二维数组中的元素不是单独的数据，而是由多个数据组成的一维数组。

在一维数组中，指向数组的指针每加 1，指针移动长度等于 1 个数组元素的大小；在二维数组中，指针每加 1，指针将移动 1 行。以数组 **a** 为例，若定义了指针数组的指针 **p**，则 **p** 初始指向数组中的第 1 行元素，若 **p+1**，则 **p** 将指向数组中的第 2 行元素。

综上，假设数组中的元素类型为 **int**，每行有 **n** 个元素，则数组指针每加 1，指针实际移动的长度为 **n*sizeof(int)**。

另外，一般用数组名与行号表示一行数据。以上面定义的数组 **a**[][] 为例，**a**[0] 表示第 1 行数据，**a**[1] 表示第 2 行数据。**a**[0] 和 **a**[1] 相当于二维数组中一维数组的数组名，指向二维数组中对应行的第 1 个元素，即 **a**[0]=&**a**[0][0]，**a**[1]=&**a**[1][0]。

已经得到二维数组中每行元素的首地址，那么该如何获取二维数组中的单个元素呢？此时，仍将二维数组视为数组元素为一维数组的一维数组，将一个一维数组视为一个元素，再单独获取一维数组中的元素。已知一维数组的首地址为 $a[i]$ ，此时的 $a[i]$ 相当于一个一维数组的数组名，类比一维数组中使用指针的基本原则，使 $a[i]+j$ ，则可以得到第 i 行中第 j 个元素的地址，对其使用“*”操作符，则 $*(a[i]+j)$ 表示二维数组中的元素 $a[i][j]$ 。若类比取值原则对地址 $a[i]$ 进行转换，则 $a[i]$ 可表示为 $a+i$ 。

需要注意一个问题，即 $a+i$ 与 $*(a+i)$ 的意义。通过前面一维数组的学习可知“*”表示取指针指向的地址存储的数据。但在二维数组中， $a+i$ 虽然指向的是该行元素的首地址，但是它代表的是整行数据元素，只是一个地址，并不表示某个元素的值。 $*(a+i)$ 仍然表示一个地址，与 $a[i]$ 等价。 $*(a+i)+j$ 表示二维数组元素 $a[i][j]$ 的地址，等价于 $\&a[i][j]$ ，也等价于 $a[i]+j$ 。

仍以数组 $a[i][j]$ 为例，表 9-1 为二维数组中相关指针与数据的表示形式及含义。

表 9-1 二维数组中相关指针与数据的表示形式及含义

表示形式	含 义
a	二维数组名，指向一维数组 $a[0]$ ，为 0 行元素首地址，也就是 $a[0][0]$ 的地址
$a[i]$ 、 $*(a+i)$	一维数组名，表示二维数组第 i 行元素首地址，值为 $\&a[i][0]$
$*(a+i)+j$	二维数组元素地址，二维数据中最小数据单元地址，等价于 $\&a[i][j]$
$*((a+i)+j)$	二维数组元素，表示第 i 行、第 j 列数据的值，等价于 $a[i][j]$

定义指向二维数组的指针时要使用数组指针，例如：

```
int array[3][3], (*p)[3];
p=array;
```

数组指针定义的一般格式如下：

变量类型 (*变量名)[数组长度]

定义数组指针时要注意，格式中的圆括号不能省略，数组指针的长度要与它将要指向的二维数组的列标相等。

上述代码 $(*p)[3]$ 中，圆括号的优先级高，首先说明 p 是一个指针，指向一个整型的一维数组，这个一维数组的长度是 3，也可以说是 p 的长度，即执行 $p+1$ 时， p 要跨过 3 个整型数据的长度，刚好指向 $array$ 数组第 2 行元素的首地址，因为 $array$ 数组每行有 3 列。

【案例 9-5】 编写程序，利用指向二维数组的指针打印杨辉三角的前 10 行。

案例分析：杨辉三角包含行和列，用二维数组来存储。某行的列数与所在行的行号相等，即第 1 行只有 1 个元素，第 2 行有 2 个元素，以此类推。每行第 1 列和最后 1 列的值为 1。从第 3 行开始，每个元素的值等于“前一行、前一列”元素的值加上“前一行、同一列”元素的值。

```
#include <stdio.h>
#define N 10
int main()
{
    int array[N][N];
    int i,j, (*p)[N];          /*定义数组指针*/
```

```

p=array;
for(i=0;i<N;i++)
    *(*(p+i))=*(*(p+i)+i)=1;          /*为每行的第1个和最后1个元素赋值*/
for(i=2;i<N;i++)
{
    for(j=1;j<i;j++)
        *(*(p+i)+j)=*(*(p+i-1)+(j-1))+*(*(p+i-1)+j);
                                          /*为第3行开始的元素赋值*/
}
for(i=0;i<N;i++)
{
    for(j=0;j<=i;j++)
        printf("%5d",*(*(p+i)+j));    /*输出每个元素*/
    printf("\n");
}
return 0;
}

```

232

程序的运行结果如图 9-8 所示。在程序中，定义了一个长度与数组列长度相同的数组指针 p，将数组首地址 array 赋给数组指针，获取第 1 行元素的首地址。指针通过*(p+i) 循环获取二维数组每行的首地址，再通过*(p+i)+j 获取每个元素的地址。在程序中，p 没有进行自增或自减操作，所以 p 都是指向第 1 行元素的首地址，可以直接用二维数组名 array 替换指针变量 p。

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
Press any key to continue

```

图 9-8 运行结果

9.3.3 字符指针

字符串本质上是以前'\0'结尾的字符数组。字符串在内存中的起始地址(即第 1 个字符的地址)通常称为字符串的指针，可以定义一个字符指针变量指向一个字符串。

1. 字符指针表示字符串

用字符指针变量 p 来表示字符串，通常有以下两种形式：

```
char *p="Good morning!";          /*边定义边赋值*/
```

或

```
char *p;
p="Good morning!";                /*先定义后赋值*/
```

用字符指针变量表示字符串，其结果是将字符串的首地址给了字符指针变量，如图 9-9 所示。

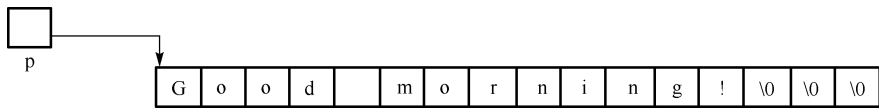


图 9-9 字符指针 p 与字符串间的关系

2. 字符串的引用

当利用字符指针变量表示字符串时，可逐个引用字符串中的字符，也可整体引用字符串。

【案例 9-6】 使用字符指针变量逐个引用字符串中的字符。

```
#include<stdio.h>
int main()
{
    char *p="I am student!";    /*初始化字符指针变量 p,将字符串的首地址赋给 p*/
    for(;*p!='\0';p++)          /*字符串结束标志'\0'作为循环结束的条件*/
        printf("%c",*p);
    return 0;
}
```

233

程序的运行结果如图 9-10 所示。

I am student!Press any key to continue_

图 9-10 运行结果

【案例 9-7】 使用字符指针变量整体引用字符串。

```
#include<stdio.h>
int main()
{
    char *p="I am student!";
    printf("%s",p); /*p 先指向字符串的第 1 个字符，然后 p 自动加 1，指向下一个字符，
                    重复这个过程，直到遇到字符串结束标志'\0'为止*/
    return 0;
}
```

程序的运行的结果与案例 9-6 的运行结果完全一致。



9.4 指针与函数

9.4.1 指针和数组名作为函数参数

1. 指针作为函数参数

在 C 语言中，实参与形参之间的数据传递是单向的值传递，即只能由实参传递给形参，

而不能由形参传递给实参，这与 C 语言中内存的分配方式有关。当发生函数调用时，系统会使用与形参对应的实参为形参赋值，此时的形参，以及该函数中的变量都存放到函数调用过程中系统在栈区开辟的空间中。栈区在函数调用时被分配，在函数调用结束时被回收。在此过程中，栈区对主调函数是不可见的，因此主调函数并不能读取栈区中形参的数据。若要将栈区的数据传递给主调函数，只能用 `return` 关键字来实现。虽然 `return` 能带回被调函数中的数据，但最多只能返回一个数据，往往不能达到要求。函数部分也曾讲到全局变量，但这种方式违背模块化程序设计的原则，与函数的设计思想背道而驰。

本节将学习一种新的方法，即使用指针变量作为函数的形参，通过传递地址的方式，使形参与实参都指向主调函数中数据所在地址，从而允许被调函数对主调函数中的数据进行操作。

指针作为函数形参的一般形式如下：

返回值类型 函数名(变量类型 *变量名, ...)

定义时，变量名前加*，表示此形参是指针类型的形参。

指针作为函数实参的一般调用形式如下：

函数名(实参变量地址, ...)

234

【案例 9-8】 两位同学一起玩猜硬币游戏：开始时，A 同学的左手握着一枚硬币，游戏开始后，A 进行有限次或真或假的交换，最后由 B 同学来猜这只手中是否有硬币，编写程序，实现游戏过程。

案例分析：因为游戏要执行有限次，所以需要首先确定交换进行的次数，通过循环执行每次交换；又因为每次交换是真是假不确定，所以至少需要实现两个交换函数，一个函数能实现两只手中硬币的交换，另一个函数只需要表面完成交换。而每次是否真正交换硬币也是随机的，因此会用随机函数发生器来决定每次选择执行的函数。

```
#include<stdio.h>
#include<stdlib.h>
void swap1(int l,int r)
{
    int t;
    t=l;l=r;r=t;
}
void swap2(int *l,int *r)
{
    int *t;
    t=l;l=r;r=t;
}
void swap3(int *l,int *r)
{
    int t;
    t=*l;*l=*r;*r=t;
}

int main()
{
```

```

int a=0,i=0,j;
int l=1,r=0;
srand((unsigned int)time(NULL));
i=5+(int)(rand()%5);
j=i;
printf("a:%d,i:%d\n",a,i);
printf("原始状态:\nl=%d,r=%d\n",l,r);
while(i>0)
{
    i--;
    a=1+(int)(rand()%3);
    switch(a)
    {
    case 1:
        swap1(l,r);
        printf("swap1:第%d次交换后的状态\nl=%d,r=%d\n\n",j-i,l,r);
        break;
    case 2:
        swap2(&l,&r);
        printf("swap2:第%d次交换后的状态\nl=%d,r=%d\n\n",j-i,l,r);
        break;
    case 3:
        swap3(&l,&r);
        printf("swap3:第%d次交换后的状态\nl=%d,r=%d\n\n",j-i,l,r);
        break;
    }
}
return 0;
}

```

程序的运行结果如图 9-11 所示。

在本案列程序中有 3 个交换函数：第 1 个函数传入的是基变量，采用的是值传递方式，函数中的数据交换随函数的结束与栈的回收而失效，并不能对主函数产生影响，所以是假交换；第 2 个函数传入的为指针变量，将两个基变量的地址作为参数传入函数，但是函数中修改的只是这两个形参指针的指向，并未修改原地址中的数据，所以同样是假交换；第 3 个函数传入指针变量，通过指针变量操作了地址中对应的变量，所以是真交换。

2. 一维数组名作为函数参数

一维数组名的值是一个指向该数组第 1 个元素的指针，当一个一维数组的数组名作为函数参数传递给另外一个函数时，实质上传递的是一份该指针的副本。函数通过这个指针的副本执行的间接访问操作，可以修改调用程序中的数组元素。

```

a:0,i:5
原始状态:
l=1,r=0
swap3:第1次交换后的状态
l=0,r=1

swap3:第2次交换后的状态
l=1,r=0

swap2:第3次交换后的状态
l=1,r=0

swap1:第4次交换后的状态
l=1,r=0

swap3:第5次交换后的状态
l=0,r=1
Press any key to continue

```

图 9-11 运行结果

【案例 9-9】 分析下面程序的执行过程。

```

#include <stdio.h>
#define N 5
void Output(int *p,int n)
{
    int *ptr;
    for(ptr = p;ptr<p+n;ptr++)
        printf("%5d",*ptr);
    printf("\n");
}
int main()
{
    int array[N] = {45,0,3,12,65};
    Output(array,N);
    return 0;
}

```

236

主函数中定义了一个具有 5 个整型数组元素的一维数组 `array`，这 5 个数组元素分别被赋值为 45、0、3、12 和 65，函数 `Output()` 实现的是数据的输出。函数 `Output()` 有两个参数：第 1 个参数是一个整型指针变量 `p`，用来接收数组的首地址；第 2 个参数是一个整型变量 `n`，用来接收数组的大小。在程序的执行过程中，主函数调用 `Output()`，将 `array` 作为实参传递给形参 `p`，即 `p=array`，将 `N` 作为实参传递给形参 `n`，即 `n=N`。函数 `Output()` 在执行的过程中，从

```

45    0    3    12   65
Press any key to continue

```

数组的第 1 个元素开始，依次输出每个元素。

程序的运行结果如图 9-12 所示。

图 9-12 运行结果

一般来说，数组名作为函数参数有以下 4 种情况。

(1) 实参与形参都用数组名

例如：

```

int main()
{
    int array[10];
    ...
    Output(array,10);
    ...
    return 0;
}

```

定义 `Output()` 函数：

```

void Output(int p[],int n)
{
    ...
}

```

形参 `int p[]` 表示 `p` 指向对象的指针变量，`int p[]` 等价于 `int *p`。

由于形参数组名接收了实参数组的首地址，因此可以理解为，在函数调用期间，形参数组与实参数组共用一段内存空间。

(2) 实参用数组名，形参用指针变量

例如：

```
int main()
{
    int array[10];
    ...
    Output (array,10);
    ...
    return 0;
}
```

定义 Output() 函数：

```
void Output(int *p,int n)
{
    ...
}
```

函数开始执行时，p 指向 array[0]，即 $p=\&\text{array}[0]$ 。通过 p 值的改变，可以指向数组 array 中的任意一个元素。

(3) 实参与形参都用指针变量

例如：

```
int main()
{
    int array[10],*ptr=array;
    ...
    Output (ptr,10);
    ...
    return 0;
}
```

定义 Output() 函数：

```
void Output(int *p,int n)
{
    ...
}
```

如果实参用指针变量，则这个指针变量必须有一个确定的值。先使实参指针变量 ptr 指向数组 array，ptr 的值是 $\&\text{array}[0]$ ，然后将 ptr 的值传给形参指针变量 p，p 的初始值也是 $\&\text{array}[0]$ 。通过 p 值的改变可以使 p 指向数组 array 的任意一个元素。

(4) 实参用指针变量，形参用数组名

例如：

```
int main()
{
    int array[10],*ptr=array;
```

```

...
Output (ptr,10);
...
return 0;
}

```

定义 Output() 函数:

```

void Output(int p[],int n)
{
    ...
}

```

实参 ptr 为指针变量, 指针变量 ptr 指向数组 array。形参为数组 p, 实际上将 p 作为指针变量处理, 可以理解为形参数组 p 和 array 数组共用同一段内存单元。在函数执行过程中可以使 p[i] 的值变化, 而它也就是 array[i]。

实参数组名代表一个固定的地址, 或者说是指针型常量, 而形参数组并不是一个固定的值。作为指针变量, 在函数调用时, 它的值等于实参数组首地址, 但在函数执行期间, 它可以再被赋值。

238

【案例 9-10】编写程序, 实现在一维数组 array[8]={68,58,64,9,72,22,52,91} 中删除元素 68。要求使用函数实现该功能, 并分别输出删除前、后的数组。

案例分析: 要在数组中删除某个元素, 就是使用其后存储空间中的元素将该存储位置上的元素覆盖, 依次使用后一个元素将前一个元素覆盖, 直至数组的最后一个元素, 最后将数组的长度减 1。

```

#include <stdio.h>
#define N 8
void delArray(int *p,int n,int x)
{
    int location=0,*ptr,i;
    for(ptr=p,i=0;ptr<p+n&& i<n;ptr++,i++)
        if(*ptr==x)
            location=i;
    for(i=location;i<n-1;i++)
        *(p+i)=*(p+i+1);
}
void Output(int *p,int n)
{
    int *ptr,i;
    for(ptr=p,i=0;ptr<p+n && i<n;ptr++,i++)
    {
        printf("%5d",*ptr);
        if((i+1)%8==0)
            printf("\n");
    }
    printf("\n");
}

```



```

int main()
{
    int array[8]={68,58,64, 9,72,22,52,91};
    printf("Del before: \n");
    Output(array,N);
    delArray(array,N,68);
    printf("Del after: \n");
    Output(array,N-1);
    return 0;
}

```

程序的运行结果如图 9-13 所示。

```

Del before:
 68  58  64   9  72  22  52  91
Del after:
 58  64   9  72  22  52  91
Press any key to continue

```

图 9-13 运行结果

3. 二维数组名作为函数参数

作为函数参数的二维数组名的传递方式和一维数组名相同，实际传递的是指向数组第 1 个元素的指针。二者之间的区别在于：二维数组的每个元素本身是另一个数组，编译器需要知道它的维数，即为函数形参的下标表达式求值。例如：

```

int array[10];
...
function1(array);

```

参数 `array` 的类型是指向整型的指针，所以 `function1()` 的原型可以是下面两种形式中的任何一种：

```

void function1(int p[ ]);
void function1(int *p);

```

作用于 `p` 上的指针运算将整型的长度作为它的调整因子。

现在来看二维数组：

```

int matrix[3][4];
...
function2(matrix);

```

参数 `matrix` 的类型是指向包含 4 个整型元素的数组的指针，所以 `function2()` 的原型可以是下面两种形式中的任何一种：

```

void function2(int (*mat)[4]);
void function2(int mat[][4]);

```

在这个函数中，`mat` 的第 1 个下标根据包含 4 个元素的整型数组的长度进行调整，接着第

2 个下标根据整型的长度进行调整，这与原先的 `matrix` 数组是一样的。

关键在于，编译器必须知道第二维的长度，才能对各下标进行求值。因此，在原型中必须声明第二维的长度。

在编写一维数组形参函数原型时，既可以把它写成数组的形式，也可以把它写成指针的形式。但是对于二维数组，只有第一维可以如此选择。若将 `function2()` 写成下面的形式是不正确的：

```
void function2(int **mat);
```

这是因为此声明把 `mat` 声明为一个指向整型指针的指针，它与指向整型数组的指针不是一回事。

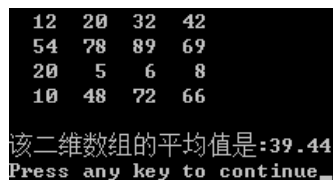
【案例 9-11】 编写一个通用函数，求 N 行 N 列的二维数组中所有元素的平均值。在主函数中使用 `matrix[4][4]={12,20,32,42,54,78,89,69,20,5,6,8,10,48,72,66}` 进行调用。

```
#include <stdio.h>
#define N 4
double avg2D(int mat[][N])
{
    double sum=0,avg;
    int i,j;
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            sum+=mat[i][j];
    avg=sum/(N*N);
    return avg;
}
int main()
{
    int matrix[N][N]={12,20,32,42,54,78,89,69,20,5,6,8,10,48,72,66};
    int i,j;
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
            printf("%4d",*(matrix+i+j));
        printf("\n");
    }
    printf("\n 该二维数组的平均值是: %.2f\n",avg2D(matrix));
    return 0;
}
```

程序的运行结果如图 9-14 所示。

9.4.2 指针作为函数的返回值

C 语言的函数可以返回除数组和函数外的任何类型数据及指向任何类型的指针，如数组指针、函数指针、void 指针，返回指针的函数称为指针函数。



```
12  20  32  42
54  78  89  69
20   5   6   8
10  48  72  66

该二维数组的平均值是:39.44
Press any key to continue_
```

图 9-14 运行结果

指针函数说明的一般形式如下:

[存储类型区分符] 类型区分符 *函数名(参数表), ...;

其中, “*函数名(参数表)”是指针函数说明符, 例如:

```
int *a(int,int);
```

a 是一个整型指针函数, 它有两个参数, 返回值是一个指向整型数据的指针。

注意, 不可将 `int *a(int,int)` 写成 `int(*a)(int,int)`, 二者说明的对象是完全不同的两个概念。后者表示 a 是一个指针变量。

【案例 9-12】 编写一个指针函数 `strstr(s,t)`, 在字符串 s 中查找子串 t, 如果找到, 返回 t 在 s 中第 1 次出现的起始位置, 否则返回 0。

```
char *strstr(char *s,char *t)    /*s 和 t 指向两个字符串*/
{
    char *ps=s,*pt,*pc;          /*ps 指向字符串 s*/
    while(*ps!='\0')              /*当 ps 没有指向字符串结束标志时,继续向后查找*/
    {
        /*如果两个指针所指字符相等,继续向后找*/
        for(pt=t,pc=ps; (*pt!='\0')&&(*pt==*pc); pt++,pc++);
        if (*pt=='\0')
            return ps; /*如果 pt 指向字符串结束标志,说明在字符串 s 中找到了
                        字符串 t, 返回 ps, ps 指向当前串 s 的某一个位置*/
        ps++;
    }
    return 0;                      /*如果查找失败, 返回 0*/
}
```

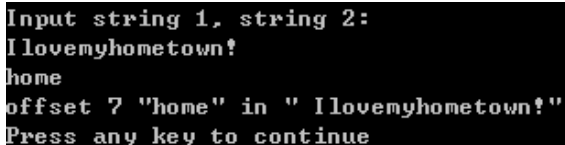
241

【案例 9-13】 编写程序, 实现输入长度不超过 50 个字符的一行正文 str 和一个字符串 substr, 求 substr 在 str 中第 1 次出现的位置。若找到, 则输出相应的信息; 否则输出未找到信息。(主函数代码如下, 子函数 `strstr` 的代码见案例 9-12。)

```
#include <stdio.h>
#define strlen 50
char *strstr(char *s,char *t);    /*指针函数 strstr 的声明*/
int main()
{
    char str[strlen],substr[strlen],*ps;    /*声明字符数组及指针变量*/
    printf("Input string 1,string 2:\n");
    scanf("%s%s",str,substr);    /*输入主串及子串*/
    ps=strstr(str,substr);        /*调用函数 strstr, 查找子串在主串中的位置*/
    if(ps!=NULL)
        printf("offset %d \"%s\" in \" %s\" \n", ps-str, substr, str);
    else
        printf("\"%s\" not exist in \"%s\" \n",substr,str);
}
```

```
    return 0;
}
```

程序的运行的结果如图 9-15 所示。



```
Input string 1, string 2:
Ilovelyhometown!
home
offset 7 "home" in " Ilovelyhometown!"
Press any key to continue
```

图 9-15 运行结果

如果一个函数返回一个指针,注意不能返回 `auto` 类型的局部变量的地址,但可以返回 `static` 类型变量的地址。例如:

```
int *returndata(int n)
{
    int m[50];
    int i;
    if(n>50) return NULL;
    for(i=0;i<n;i++)
        scanf("%d",&m[i]);
    return m;
}
```

是错误的,正确的代码如下:

```
int *returndata(int n)
{
    static int m[50];
    int i;
    if(n>50) return NULL;
    for(i=0;i<n;i++)
        scanf("%d",&m[i]);
    return m;
}
```

这是因为 `auto` 类型的局部变量的生存期很短,当函数返回时,返回的指针对应的内存单元将被释放,则返回的指针就会无效。但对于 `static` 类型的局部变量来说,因为其生存期等同于全局变量的生存期,故函数返回时,返回的指针对应的内存单元不会被释放,返回的指针也是有效的。

使用指针函数时一定要注意其返回值,避免返回的指针对应的内存空间因该指针函数的返回而被释放。

返回的指针通常有以下几种:

- ① 函数中动态分配的内存(通过 `malloc()` 函数等实现)的首地址;
- ② 通过指针形参获得的实参的有效地址;
- ③ 函数中的静态变量或全局变量对应的存储单元的首地址。

9.5 链表

9.5.1 链表的概念

链表是一种常用的，动态进行存储分配的数据结构。它与数组不同，不必事先确定元素的个数，可以根据当时的需要来开辟内存单元。它的各个元素不要求顺序存放，因此，可克服使用数组存放数据的一些不足。

由于需要动态申请内存空间，因此，以下内容将介绍申请空间常用的函数，包括：`malloc()` 函数、`calloc()` 函数、`realloc()` 函数和 `free()` 函数。它们都包含在头文件 `stdlib.h` 中。

1. 动态内存分配函数 `malloc()`

`malloc()` 函数用于申请指定大小的存储空间，函数原型如下：

```
void *malloc(unsigned int size);
```

功能：在内存的动态存储区中分配一个连续空间，其长度为 `size`。如果申请成功，则返回一个指向所分配内存空间的起始地址的指针，否则返回 `NULL` (值为 0)。函数 `malloc()` 的返回值为 `(void *)` 类型 (这是通用指针的一个重要用途)。在具体的使用中，将函数 `malloc()` 的返回值转换为特定指针类型，赋给一个指针。

调用形式如下：

```
(类型说明符*) malloc(size);
```

“类型说明符”表示把该区域用于何种数据类型。“`(类型说明符*)`”表示把返回值强制转换为该类型指针。“`size`”是一个无符号数，例如：

```
pc=(char*) malloc(100);
```

表示分配 100 个字节的内存空间，并强制转换为字符指针类型，然后把该指针赋予指针变量 `pc`。通常采用以下方式调用该函数：

```
int size=50;
int *p = (int*)malloc(size*sizeof(int));
if(p==NULL)
{
    printf("Not enough space to allocate!\n");
    exit(-1);
}
```

在调用函数 `malloc()` 时，最好利用运算符 `sizeof()` 来计算存储块的大小，不要直接写整数，因为不同平台的数据类型占存储空间的大小可能不同，每次动态分配都必须检查是否成功。此外，虽然存储空间是动态分配的，但它的大小在分配后也是确定的，需注意不可越界使用。

2. 动态内存调整函数 realloc()

函数原型如下:

```
void *realloc(void *ptr,unsigned size)
```

功能: 更改以前的存储分配空间。**ptr** 必须是以前通过动态存储分配得到的指针, 参数 **size** 为现在需要的存储空间的大小。如果调整失败, 则返回 **NULL**, 同时原来 **ptr** 指向存储空间的内容不变; 如果调整成功, 则返回一个存储大小为 **size** 的存储空间, 并保证该空间的内容与原存储空间一致。如果 **size** 小于原存储空间的大小, 则内容为原存储空间前 **size** 范围内的数据; 如果新存储空间更大, 则原有数据存储在新存储空间的前一部分。如果分配成功, 则原存储空间的内容就可能改变, 因此不允许再通过 **ptr** 使用它。

3. 计数动态存储分配函数 calloc()

函数原型如下:

```
void *calloc(unsigned n,unsigned size)
```

功能: 在内存的动态存储空间中分配 **n** 个连续空间, 每个存储空间的长度为 **size**, 并且在分配后还将存储空间全部初始化为 **0** 值。如果申请成功, 则返回一个指向被分配存储空间的起始地址的指针, 否则返回 **NULL** (值为 **0**)。

4. 动态存储释放函数 free()

函数原型如下:

```
void free(void *ptr)
```

功能: 释放 **ptr** 指向的一块内存空间。**ptr** 是一个任意类型的指针变量, 它指向被释放区域的首地址。被释放区应是由 **malloc()** 函数分配的区域。调用形式如下:

```
free(ptr);
```

【案例 9-14】 集合 **array1** 的值为 {12,45,69,7,10}, 现在要将数据 100 追加到该集合中, 编写程序实现该过程。输出追加前、后集合中的数据。

案例分析: 在为 **array1** 分配空间时, 可以使用 **malloc()** 函数进行分配, 以便使用 **realloc()** 进行扩充。

```
#include <stdio.h>
#include <stdlib.h>
#define N 5
int main()
{
    int *array1=(int*)malloc(N*sizeof(int));
    int i;
    *(array1)=12;
    *(array1+1)=45;
    *(array1+2)=69;
    *(array1+3)=7;
    *(array1+4)=10;
```

```
printf("追加前的集合 array1: \n");
for(i=0;i<N;i++)
    printf("%5d",*(array1+i));
printf("\n");
array1= (int *)realloc(array1,(N+1)*sizeof(int));
if(array1==NULL)
{
    printf("Not enough space to allocate!\n");
    exit(-1);
}
*(array1+N) = 100;
printf("追加后的集合 array1: \n");
for(i=0;i<=N;i++)
    printf("%5d",*(array1+i));
printf("\n");
free(array1);
return 0;
}
```

程序的运行结果如图 9-16 所示。

9.5.2 链表的基本操作

链表是由若干个称为结点的元素构成的。每个结点包含数据字段和链接字段。数据字段用来存放结点的数
据项；链接字段用来存放该结点指向另一个结点的指针。每个链表都有一个“头指针”，用来存放该链表的起始地址，即指向该链表的起始结点，是识别链表的标志。对某个链表进行操作，首先要知道该链表的头指针。链表的最后一个结点称为“表尾”，它不再指向任何后继结点，表示链表的结束。

链表可分为单向链表和双向链表。两者的区别仅在于结点的链接字段中，单向链表仅有一个指向后继结点的指针，而双向链表则有两个指针，一个指向后继结点，另一个指向前驱结点。图 9-17 给出单向链表与双向链表的区别。

```
追加前的集合 array1:
12  45  69  7  10
追加后的集合 array1:
12  45  69  7  10 100
Press any key to continue
```

图 9-16 运行结果

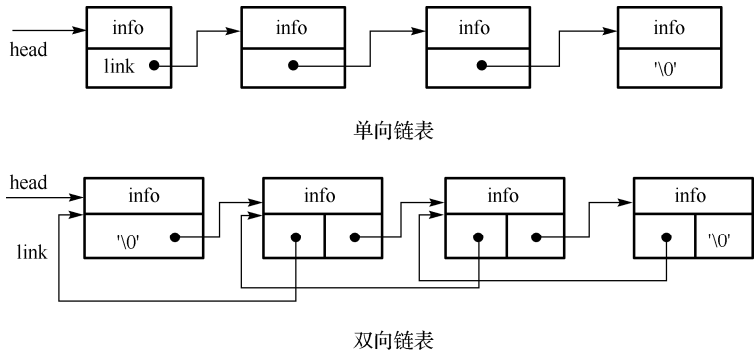


图 9-17 单向链表与双向链表的区别

链表的数据结构必须用指针来实现，每个结点中的链接字段要包含一个或两个指针，存放链接的结点地址。下面是单向链表和双向链表中结点的结构代码。

```
struct link1{
    char c1[100];
    struct link1 *next;
};
struct link2{
    char c2[100];
    struct link2 *next;
    struct link2 *prior;
};
```

其中，link1 是单向链表中结点的结构体名，link2 是双向链表中结点的结构体名。链表中的结点就是具有这种结构体名的结构体变量。这里仅是一个例子，在实际应用中，结点结构体的数据字段更加复杂。

单向链表比较简单，本节只讨论单向链表的操作。

使用一个学生成绩表，该表是由若干位学生的成绩组成的，每个结点是一位学生的成绩。为了简化结点，突出链表的操作，定义结点的结构体如下：

246

```
struct student
{
    char name[20];
    int num;
    int score;
    struct student *next;
};
```

该结构体中，有三个成员作为数据字段，其中学号 num 作为关键字，自身引用的指针作为链接字段，用来指出下一个结点的位置。为了简化结构体内容，只选用了一门课程的成绩。

1. 链表的建立

建立链表函数的算法如下。

设置三个结构指针 head、p 和 q。先用 malloc() 开辟一个结点，并且使指针 p 和 q 分别指向开辟的结点。再从键盘读入一位学生的数据，赋给 p 指向的结点。假设学号 num 的值为 0 时，链表建立结束，该结点不链入表中。输入第 1 个结点的数据后，将 p 赋给 head，使 head 指向链表中的第 1 个结点，如图 9-18 所示。接着，再开辟第 2 个结点，使 p 指向新结点，读入新结点的数据，并链入新结点，将 p 的值赋给 q->next，使第 1 个结点的 next 成员指向第 2 个结点。接着，使 q=p，即 q 将指向新结点，如图 9-19 所示。

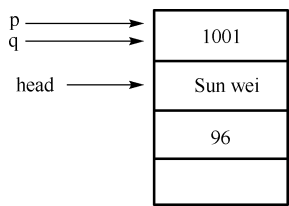


图 9-18 链表第 1 个结点的实现

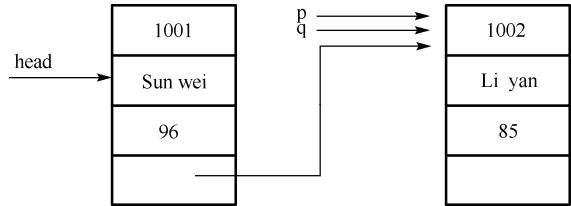


图 9-19 链表第 2 个结点的实现

再用 p 去开辟新结点, 赋值后, 在学号 num 成员不为 4 的情况下, 链入新结点。以此类推, 直到链入最后一个新结点为止。这里, 设置的三个结构指针, 除 $head$ 作为该链表的头指针之外, p 总是用来指向开辟的新结点的指针, 而 q 总是指向 p 所指向的新结点的前一个结点的指针, 通过将 p 赋值给 $q \rightarrow next$ 来链入新结点。

建立链表函数代码如下:

```
struct student*creat list()
{
    struct student *head,*p,*q;
    head=NULL;
    p=q=(struct student *)malloc(sizeof(student));
    n=0;
    printf("Input node data:");
    scanf("%d%s%d", &p->num, p->name, &p->score);
    while(p->num!=0)
    {
        n++;
        if(n==1)
        {
            head=p;
            head->next=NULL;
        }
        else
            q->next=p;
        q=p;
        p=(struct student*)malloc(sizeof(struct student));
        printf("Input node data:")
        scanf("%ld%s%d", &p->num, p->name, &p->score);
    }
    q->next=NULL;
    return(head);
}
```

247

2. 链表的删除

链表的删除操作就是将一个待删除结点从链表中分离出来, 不再与链表的其他结点有任何联系。为了从链表中删除一个结点, 需要考虑如下 4 种情况。

- ① 如果链表为空表, 则无须删除结点, 直接退出程序即可。
- ② 如果找到待删除的结点, 而且它是首结点, 那么只要将 $head$ 指向该结点的下一个结点, 即可删除该结点, 如图 9-20 所示。

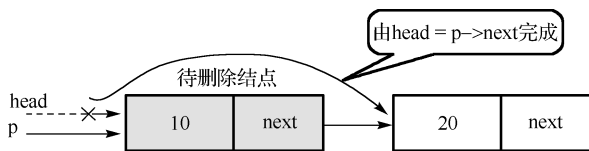


图 9-20 待删除结点是首结点的删除过程

③ 如果找到待删除的结点,但它不是首结点,那么只要将前一个结点的指针指向当前结点的下一个结点,即可删除当前结点,如图 9-21 所示。如果待删除结点是最后一个结点,则按图 9-21 操作时,由于 $p \rightarrow \text{next}$ 的值为 NULL,经 $pr \rightarrow \text{next} = p \rightarrow \text{next}$ 赋值后, $pr \rightarrow \text{next}$ 的值也为 NULL,从而使 pr 所指的结点由倒数第 2 个结点变成最后一个结点。

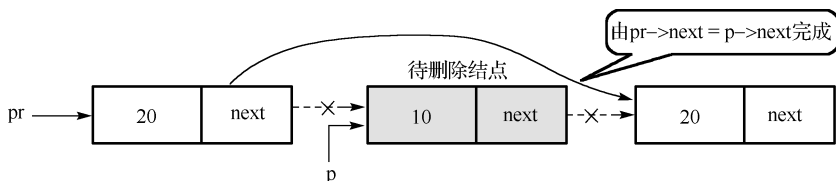


图 9-21 待删除结点不是首结点的删除过程

④ 如果已搜索到链表末尾($p \rightarrow \text{next} == \text{NULL}$),仍未找到待删除结点,则显示“未找到”。用函数 `DeleteNode()` 实现这一操作,编写程序代码如下:

```
struct student *DeleteNode(struct student *head, int stunum)
{
    struct student *p = head, *pr = head;
    if (head == NULL)           /*链表为空,没有结点,无法删除结点*/
    {
        printf("No Linked Table!\n");
        return(head);
    }
    while (stunum != p->num && p->next != NULL)
    {
        /*若没找到结点 num,且未到表尾,则继续找*/
        pr = p;
        p = p->next;
    }
    if (stunum == p->num)        /*若找到结点 num,则删除该结点*/
    {
        if (p == head)          /*若待删除结点为首结点,则让 head 指向第 2 个结点*/
        {
            head = p->next;      /*注意此时链表的头结点指针发生了改变*/
        }
        else /*若待删除结点不是首结点,则将前一个结点的指针指向当前结点的下一个结点*/
        {
            pr->next = p->next;
        }
        free(p);                /*释放为已删除结点分配的内存*/
    }
    else /*没有找到待删除结点*/
    {
        printf("This Node has not been found!\n");
    }
    return head;                /*返回删除结点后的链表的头结点指针*/
}
```

3. 链表的插入

链表的插入操作就是将一个待插入的结点插入到已经建立好的链表中的适当位置。向链表中插入一个新结点需要考虑以下4种情况。

① 若原链表为空表，则新插入结点作为首结点，让 `head` 指向新插入结点 `p`，且置新结点的指针域为空 (`p->next=NULL`) 即可。

② 若按结点数据的排序结果在首结点前插入新结点，则将 `head` 指向新结点 `p`，而新结点的指针域指向原来链表的头结点 (`p->next=head`)，如图 9-22 所示。

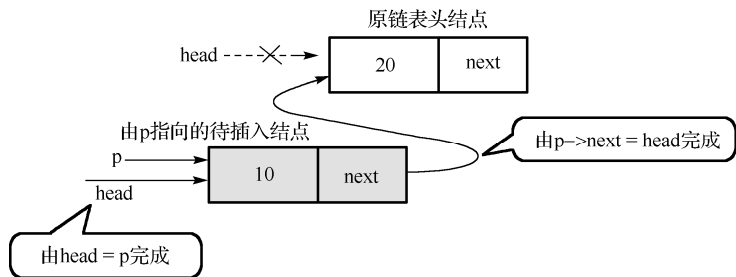


图 9-22 在首结点前插入新结点的过程

③ 若按结点数据的排序结果在链表中间插入新结点，则将待插入结点 `p` 的指针域指向下一个结点 (`p->next=pr->next`)，而前一个结点的指针域指向待插入结点 `p` (`pr->next=p`)，操作方法如图 9-23 所示。

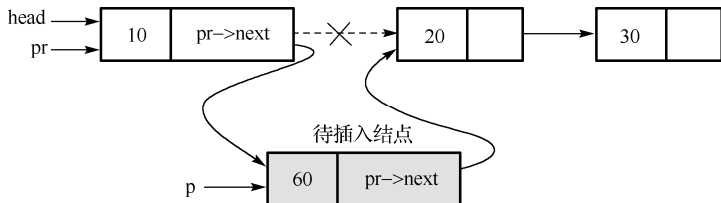


图 9-23 在链表中间插入新结点的过程

④ 若按结点数据的排序结果在链表末尾插入新结点，则将链表的最后一个结点的指针域指向待插入结点 `p` (`pr->next=p`)，而待插入结点的指针域置 `NULL`，如图 9-24 所示。

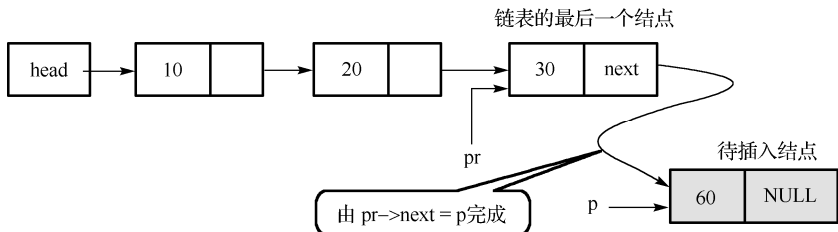


图 9-24 在链表末尾插入新结点的过程

使用函数 `InsertNode()` 实现向结点数据按升序排序的链表中插入一个新结点，程序代码如下：

```
struct student *InsertNode(struct student *head, int stunum)
{
    struct student *pr = head, *p = head, *temp = NULL;
```

```
p = (struct student *)malloc(sizeof(struct student));
/*为新插入结点申请内存*/
if (p == NULL)
/*如果申请内存失败，则退出程序*/
{
    printf("No enough memory!\n");
    exit(0);
}
p->next = NULL;
/*置新结点的指针域赋为空*/
p->num = stunum;
/*为新结点数据域赋值 stunum*/
if (head == NULL)
/*若原链表为空表，则新插入结点作为首结点*/
{
    head = p;
}
else
/*若链表为非空*/
{
    /*若没找到待插入结点的位置，则继续找*/
    while (pr->num < stunum && pr->next != NULL)
    {
        temp = pr;
        pr = pr->next;
    }
    if (pr->num >= stunum)
    {
        if (pr == head)
/*在首结点前插入新结点*/
        {
            p->next = head;
            head = p;
/*注意此时链表的头结点指针发生了改变*/
        }
        else
/*在链表中间插入新结点*/
        {
            pr = temp;
            p->next = pr->next;
            pr->next = p;
        }
    }
    else
/*在表尾插入新结点*/
    {
        pr->next = p;
    }
}
return head;
/*返回插入新结点后的链表的头结点指针*/
}
```



9.6 本章小结

指针是 C 语言重要的组成部分。本章通过多个案例，讲解了指针与指针变量的概念、指针变量的引用、指针的运算、字符指针、数组指针的定义与使用方法，还讲解了如何使用指

针引用一维数组与二维数组，以及如何分配和回收内存等。通过本章的学习，读者应掌握多种指针的定义与使用方法，使用指针优化程序代码，提高程序代码的灵活性。

9.7 习题

一. 单选题

1. 若有定义“int x,*pb;”，则以下表达式正确的是()。

- A. pb=&x B. pb=x C. *pb=&x D. *pb=*x

2. 以下程序的输出结果是()。

```
#include<stdio.h>
void main()
{
    printf("%d\n",NULL);
}
```

- A. 因变量无定义，输出不定值 B. 0
C. -1 D. 1

3. 以下程序的输出结果是()。

```
#include<stdio.h>
void sub(int x,int y,int *z)
{
    *z=y-x;
}
void main()
{
    int a,b,c;
    sub(10,5,&a); sub(7,a,&b); sub(a,b,&c);
    printf("%d,%d,%d\n",a,b,c);
}
```

- A. 5,2,3 B. -5,-12,-7 C. -5,-12,-17 D. 5,-2,-7

4. 以下程序的输出结果是()。

```
#include<stdio.h>
void main()
{
    int k=2,m=4,n=6;
    int *pk=&k, *pm=&m, *p;
    *(p=&n)=*pk*( *pm);
    printf("%d\n",n);
}
```

- A. 4 B. 6 C. 8 D. 10

5. 已知指针 p 指向下图中的 a[1]，则执行语句“*p++；”后，*p 的值是()。

10	a[0]
20	a[1]
30	a[2]
40	a[3]
50	a[4]

- A. 20 B. 30 C. 21 D. 31

6. 已知指针 p 指向下图中的 a[1]，则表达式 “*++p” 的值是()。

10	a[0]
20	a[1]
30	a[2]
40	a[3]
50	a[4]

- A. 20 B. 30 C. 21 D. 31

7. 已知指针 p 指向下图中的 a[1]，则表达式 “++*p” 的值是()。

10	a[0]
20	a[1]
30	a[2]
40	a[3]
50	a[4]

- A. 20 B. 30 C. 21 D. 31

8. 以下程序的输出结果是()。

```
#include<stdio.h>
void prtv(int *x)
{printf("%d\n",++*x);}
void main()
{
    int a=25;prtv(&a);
}
```

- A. 23 B. 24 C. 25 D. 26

9. 以下程序的输出结果是()。

```
#include<stdio.h>
void main()
{
    int **k,*a,b=100;
    a=&b;k=&a;
    printf("%d\n",**k);
}
```

- A. 运行出错 B. 100 C. a 的地址 D. b 的地址

10. 以下程序的输出结果是()。

```
#include<stdio,h>
ss(char *s)
{   char *p=s;
    while(*p) p++;
    return(p-s);
}
main()
{   char *a="abded";
    int i;
    i=ss(a);
    printf("%d\n",i);
}
```

A. 8 B. 7 C. 6 D. 5

11. 在下面程序横线处填入语句，使程序的运行结果与其余结果不同的是()。

```
#include<stdio.h>
void main(void)
{
    int a[5]={1,3,5,7,9},*p,i;
    p=a;
    for(i=0;i<5;i++)
        printf("%4d",_____);
}
```

A. *(p+i) B. *p+i C. *(p++) D. *p++

12. 以下程序的输出结果是()。

```
#include <stdio.h>
struct NODE
{
    int k;
    struct NODE *link;
};
void main(void)
{
    struct NODE m[5],*p=m,*q=m+4;
    int i=0;
    while(p!=q)
    {
        p->k=++i; p++;
        q->k=i++; q--;
    }
    q->k=i;
    for(i=0;i<5;i++)
        printf("%d",m[i].k);
    printf(" ");
}
```

A. 12321 B. 13431 C. 02442 D. 12345

13. 如果将第 12 题程序中语句 “p->k=++i;” 改成 “p->k=i++;”, 将语句 “q->k=i++;” 改成 “q->k=++i;”, 则程序的运行结果是()。

A. 12321 B. 13431 C. 02442 D. 12345

14. 已知有声明 “int a[3][3]={0},*p1=a[1],(*p2)[3]=a;”, 以下表达式中与 “a[1][1]=1” 不等价的表达式是()。

A. *(p1+1)=1 B. p1[1][1]=1; C. (*(p2+1)+1)=1 D. p2[1][1]=1;

15. 假如指针 p 已经指向某个整型变量 x, 则&*p 相当于()。

A. &x B. x C. *p D. **p

16. 以下程序的运行结果是()。

```
#include <stdio.h>
union ks
{
    int a;
    int b;
} *p, s[4]={0};
void main(void)
{
    int n=1, i;
    for(i=1; i<4; i++)
    {
        s[i].b=s[i-1].a+1;
        s[i].a=s[i-1].b+n;
        n=n+2;
    }
    p=&s[3];
    printf("%d\n", p->a);
}
```

A. 3 B. 6 C. 9 D. 12

17. 如果将第 16 题程序中 “s[i].b=s[i-1].a+1;s[i].a=s[i-1].b+n;” 两条语句改写为 “s[i].a=s[i-1].b+n;s[i].b=s[i-1].a+1;”, 则程序的运行结果是()。

A. 3 B. 6 C. 9 D. 12

18. 若有定义 “char *x="abcdefghi";”, 则以下选项中正确运用 strcpy() 函数的是()。

A. char y[10],*s; strcpy(s=y+1,x+1); B. char y[10],*s; strcpy(s=y+5,x);
C. char y[10]; strcpy(++y,&x[1]); D. char y[10]; strcpy(y,x[4]);

19. 若有定义 “int a[3][4];”, 则不能表示数组元素 a[1][1] 的是()。

A. *(a[1]+1) B. *(a+5) C. (*(a+1)[1]) D. *(&a[1][1])

20. 若有语句 “char s[10],*p=s;”, 则下面表达式不正确的是()。

A. p=s+5; B. s=p+s; C. s[2]=p[4]; D. p=s[0];

21. 若有语句 “int a[]={0,1,2,3,4},i,*p=a;”, 其中 $0 \leq i < 5$, 则以下不表示数组元素的是()。

A. *(a+i) B. p+i C. a[p-a] D. *(&a[i])

22. 若有语句“int x[5][6],*p=*x;”,则下面不表示数组元素 x[0][2]的是()。
- A. p[0][2] B. *x+2 C. *(*x+2) D. x[0][2]
23. 若有语句“int (*p)[m];”,则 p 是()。
- A. m 个指向整型变量的指针
B. 指向 m 个整型变量的函数指针
C. 一个指向具有 m 个整型元素的一维数组的指针
D. 具有 m 个指针元素的一维指针数组,每个元素只能指向整型变量
24. 以下程序的输出结果是()。

```
char str[ ]="ABCD";
char *p;
for(p=str;p<str+4;p++)
printf("%s\n",str);
```

- A. ABCD B. A C. D D. ABCD
- B C BCD
- C B CD
- D A D

二、填空题

1. 若有定义“char ch;”,则:

- (1)使指针 p 指向变量 ch 的定义语句是_____。
- (2)若定义“char *p;”,使指针 p 指向变量 ch 的赋值语句是_____。
- (3)在(1)的基础上,通过指针 p 给变量 ch 读入字符的 scanf 调用语句是_____。
- (4)在(1)的基础上,通过指针 p 给变量 ch 赋字符 a 的语句是_____。
- (5)在(1)的基础上,通过指针 p 用格式输出函数输出 ch 中字符的语句是_____。
2. 以下程序输出的是_____。

```
void main()
{   int i=3, j=2;
    char *a="DCBA";
    printf("%c%c\n",a[i],a[j]);
}
```

3. 以下程序用来计算一个英文句子中最长的单词的长度 max。假设句子中只有字母和空格,单词之间用空格分隔,句子以“.”结束。请在程序的对应位置填空。

```
#include <stdio.h>
void main(void)
{
    char *p,a[]="This is an example.";
    int max=0,k=0;
    _____;
    while(*p!='.')
    {
```

```

        while(*p>='a'&&*p<='z' || *p>='A'&&*p<='Z')
        {
            _____;
            p++;
        }
        if(k>max)
            _____;
        k=0;
        p++;
    }
    printf("Max=%d\n",max);
}

```

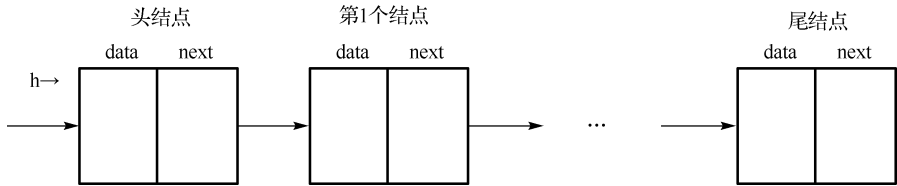
4. 函数 `deletelist()` 的功能为：在 `head` 指向的单项链表中查找是否出现多个与 `x` 值相同的结点。如果发现存在这样的结点，则保留第 1 个结点，删除其他重复出现的结点。请在程序的对应位置填空。

```

#include<stdlib.h>
typedef struct point
{
    int x;
    struct point *next;
}List;
List *deletelist (List *head)
{
    List *p,*p1,*p2;
    p=_____ ;
    while(p->next!=NULL)
    {
        p1=p;
        p2=p->next;
        while(p2!=NULL)
        {
            if(p2->x==p->x)
            {
                p1->next=_____ ;
                free(p2);
                p2=p1->next;
            }
            else
            {
                p1=p2;
                p2=p2->next;
            }
        }
        p=_____ ;
    }
    return head;
}

```

5. 以下 creat() 函数的功能是：建立一个如下图所示的带头结点的单向链表，要求链表新产生的结点总是插在链表的末尾处，单向链表的头指针作为函数值返回。请在程序的对应位置填空。



```

#include<stdio.h>
#include<stdlib.h>
struct list
{
    int data;
    struct list *next;
};
struct list *creat()
{
    struct list *h,*p,*q;
    int dat;
    h=(struct list *)malloc(sizeof(struct list));
    p=q=h;
    scanf("%d",&dat);
    while(dat!=0)
    {
        p=(struct list *)malloc(sizeof(struct list));
        p->data=_____ ;
        q->next=p;
        _____ ;
        scanf("%d",&dat);
    }
    p->next=NULL;
    return h;
}
void main()
{
    struct list *h,*p;
    h=creat();
    p=_____ ;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}
    
```

6. 以下程序的功能是：统计带有头结点的单向链表中结点的个数，存放在形参 *n* 所指的存储单元中。请在程序的对对应位置填空。

```
#include<stdio.h>
#include<stdlib.h>
#define N 8
typedef struct list
{
    int data;
    struct list *next;
}SLIST;
SLIST *creatlist(int *a);
void outlist(SLIST *);
void fun(SLIST *h,int *n)
{
    SLIST *p;
    _____=0;
    p=h->next;
    while(p)
    {
        (*n)++;
        p=_____;
    }
}
void main(void)
{
    SLIST *head;
    int a[N]={12,65,45,32,70,16,20,48},num;
    head=creatlist(a);
    outlist(head);
    fun(_____, &num);
    printf("\nnumber=%d\n",num);
}
SLIST *creatlist(int a[])
{
    SLIST *h,*p,*q;
    int i;
    h=p=(SLIST *)malloc(sizeof(SLIST));
    for(i=0;i<N;i++)
    {
        q=(SLIST *)malloc(sizeof(SLIST));
        q->data=a[i];
        p->next=q;
        p=q;
    }
    p->next=0;
    return h;
}
void outlist(SLIST *h)
{
    SLIST *p;
```

```

    p=h->next;
    if(p==NULL)
        printf("The list is NULL!\n");
    else
    {
        printf("\nHead ");
        do
        {
            printf("->%d",p->data);
            p=p->next;
        }while(p!=NULL);
        printf("->End\n");
    }
}

```

7. 以下程序的功能是：通过指针操作，找出三个整数中的最小值，并输出。请在程序的对应位置填空。

```

#include "stdio.h"
main()
{
    int *a,*b,*c,num,x,y,z;
    a=&x;b=&y;c=&z;
    printf("输入 3 个整数: ");
    scanf("%d%d%d",a,b,c);
    printf("%d,%d,%d\n",*a,*b,*c);
    num=*a;
    if(*a>*b)_____ ;
    if(num>*c)_____ ;
    printf("输出最小整数:%d\n",num);
}

```

8. 下面程序的运行结果是_____。

```

chars[80],*sp="HELLO!";
sp=strcpy(s,sp);
s[0]='h';puts(sp);

```

9. 下面程序的运行结果是_____。

```

char str[]="abc\0def\0ghi",*p=str;
printf("%s",p+5);

```

10. 下面程序的功能是：将两个字符串 s1 和 s2 连接起来。请在程序的对应位置填空。

```

#include<stdio.h>
main()
{
    char s1[80],s2[80]; gets(s1); gets(s2);
    conj(s1,s2); puts(s1);
}
void conj(char *p1,char *p2)
{

```

```

char *p=p1;
while(*p1)_____ ;
while(*p2){*p1=_____ ;p1++;p2++;}
*p1='\0';
_____ ;
}

```

11. 若有定义 “int a[]={2,4,6,8,10,12},*p=a;”, 则*(p+1)的值是_____, *(a+5)的值是_____。
12. 若有定义 “int a[2][3]={2,4,6,8,10,12};”, 则 a[1][0]的值是_____, *((a+1)+0)的值是_____。

三、上机操作题

- 编写程序, 计算传递来的两个整型量的和与积, 并通过参数返回。
- 编写程序, 将用户输入的字符串中的所有数字提取出来。编写程序, 计算字符串的串长。
- 编写程序, 将一个字符串中的字母全部转换为大写。
- 编写程序, 计算一个字符在一个字符串中出现的次数。
- 编写程序, 判断一个子字符串是否在某个给定的字符串中出现。
- 编写程序, 建立一个带有头结点的单向链表, 链表结点中的数据通过键盘输入。当输入数据为-1 时, 表示输入结束(链表的头结点的 data 域不放数据, 表空的条件是 ph->next==NULL)。
- 用指针方法编写程序, 从键盘上输入一个字符串, 字符按从小到大的次序排列并输出。
- 编写 fun() 函数, 该函数的功能是: 将数组的每个元素依次往后移动一个位置, 将最后一个元素放到第 1 个位置。编写 main() 函数, 输入一个数字 (1<=n<=9), 调用 fun() 函数依次移动 n 次, 并最终回到数组原始位置。如输入为 5, 则输出结果如下所示:

```

Please input n: 5
1 2 3 4 5
Change position:
5 1 2 3 4
4 5 1 2 3
3 4 5 1 2
2 3 4 5 1
1 2 3 4 5

```

9. 编写 fun() 函数, 该函数的功能是: 统计形参 str 指向的字符串中数字字符的个数, 并将统计结果存放在数组 a 中(数字字符 0 的个数存在 a[0]中, 数字字符 1 存在 a[1]中, 以此类推)。编写 main() 函数, 输入一个字符串, 调用 fun() 函数, 并输出哪些数字字符在字符串中出现过, 以及出现的次数。例如, 输入字符串 3k4h55k6h4kj63k4h, 则输出结果如下所示:

```

3      2
4      3
5      2
6      2

```

第 10 章 文 件

教师工资管理系统中的教师工资信息在系统运行时，保存在结构体数组中；在系统没有运行时，保存在某个文件中。若每次运行系统时，前次的数据丢失，那么重新输入将降低数据的运行效率。因此，解决数据的存储问题，系统才能更加完善。教师工资管理系统中的文件需要进行打开与关闭，读与写的操作等。本章主要讲解 C 语言文件的操作。

学习目标

- 了解计算机中流和文件的概念
- 了解缓冲文件的作用
- 掌握如何使用文件指针引用文件
- 掌握文件位置指针的使用方式
- 掌握文件的打开与关闭，读与写的操作

10.1 文件概述

10.1.1 文件的概念

1. 数据流

C 语言中，将在不同输入/输出设备(键盘、显示器等)之间进行传递的数据抽象为“流”，称为数据流。例如，当调用 `scanf()` 函数时，会有数据经过键盘流入内存；当调用 `printf()` 函数时，会有数据从内存流向屏幕。流实际上是一个字节流，输入时，称为输入流；输出时，称为输出流。

根据数据形式的不同，输入流和输出流可以细分为文本流(字符流)和二进制流。文本流和二进制流之间的主要差异是：在文本流中输入、输出的数据是字符或字符串，可以修改；在二进制流中输入、输出的是一系列二进制的 0、1 代码，不能以任何形式修改。

2. 文件

文件是一组存储在外部设备上的数据的集合。这个数据集有一个名称，即文件名。文件名由路径、文件名主干和扩展名(后缀)三部分组成。计算机系统包括文件系统，按文件名对文件进行组织，以及存、取管理。图 10-1 为一个文件名，由它可以找到“D:\C 语言案例教程\demo”路径下，扩展名为“.dat”，文件名主干为“Salary”的二进制文件。

计算机中的文件可分为两类：一类为 ASCII 文件，另一类为二进制文件。

D:\C 语言案例教程\demo\Salary.dat		
路径	文件名主干	扩展名

图 10-1 文件名

ASCII 文件又称为文本文件，它将每个字节转换成与其对应的一个 ASCII 代码，然后进行存放，代表一个字符；二进制文件是把内存中的数据按其在内存中的存储形式原样输出到磁盘上存放。例如，要表示整数 322，ASCII 文件形式如图 10-2 所示，二进制文件形式如图 10-3 所示。

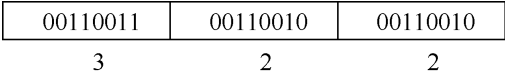


图 10-2 ASCII 文件形式



图 10-3 二进制文件形式

从图 10-2 可以看出，用 ASCII 文件形式表示整数 322，占用 3 个字节，1 个字节表示 1 个 ASCII 文件字符。输出时，对每个字符逐个进行处理，便于输出。但将二进制转换为 ASCII 码需花费时间较长，且占存储空间较多。从图 10-3 可以看出，用二进制文件形式表示整数 322，占用 2 个字节，节省存储空间。无须进行转换，节省时间。但 1 个字节不对应 1 个字符，不能按字符形式直接输出。

C 语言将文件视为一个字节序列，即文件由若干个字符(字节)数据顺序组成，也称为“流”。以字节为单位存、取，用程序控制输入、输出的数据流的开始和结束不受符号(如回车换行符)控制，将这种形式的文件称为流式文件。也就是说，C 语言中的文件并不是由记录组成的，而是由是一个字节流或二进制流组成的。

262

10.1.2 缓冲文件系统与非缓冲文件系统

C 语言使用的磁盘文件系统有两种：一种是缓冲文件系统，也称标准文件系统；另一种是非缓冲文件系统。缓冲文件系统指系统自动地在内存区为每个正在使用的文件开辟一个缓冲区。从外部介质向内存读入数据时，从磁盘文件将一些数据输入内存缓冲区(充满缓冲区)，然后再从缓冲区逐个地将数据送给接收程序变量。向外部介质写数据时，程序先将数据写入内存缓冲区，等到一定的时候才把缓冲区中的内容一次性地写入外部介质。确定缓冲区的大小，一般 C 语言版本为 512 字节。非缓冲文件系统指由用户自己根据需要为每个文件设定缓冲区，不由系统自动设置。ANSI C 只采用缓冲文件系统，因此本章只介绍缓冲文件系统的相关操作。

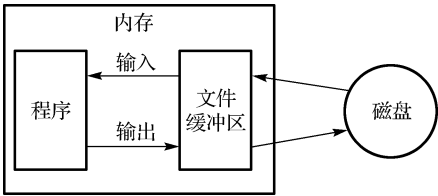


图 10-4 文件缓冲区读、写文件的过程

大小，一般 C 语言版本为 512 字节。非缓冲文件系统指由用户自己根据需要为每个文件设定缓冲区，不由系统自动设置。ANSI C 只采用缓冲文件系统，因此本章只介绍缓冲文件系统的相关操作。

使用缓冲文件系统可以减少磁盘的读、写次数，提高读、写效率，加快文件访问的速度。文件缓冲区读、写文件的过程如图 10-4 所示。

10.1.3 文件指针

文件缓冲区是内存中的一片区域，想要访问此区域中的文件，需要一个变量获取文件在内存中的地址。因此，此变量应是指针变量，且是一个文件指针。C 语言在缓冲区文件系统中定义了一个“文件指针”，它是系统定义的结构体类型，并取名为 FILE，也称 FILE 指针。通常用 FILE 类型来定义指针变量，需要多少个文件，就定义多少个变量。

定义文件指针变量的一般格式如下：


```
FILE *变量名;
```

例如：

```
FILE *fp, *np, *tp;
```

定义了 fp、np、tp 三个指针变量，都是指向 FILE 类型的指针变量。

10.2 文件的打开与关闭

C 语言主要通过标准 I/O 函数操作文件。相关的操作包括：打开与关闭、读与写，以及设置缓冲区。在对文件的读与写前，需要先打开文件；读与写结束后，应及时关闭文件。

10.2.1 打开文件函数

C 语言提供一个专门用于打开文件的库函数：fopen() 函数。fopen() 函数打开文件时先将文件复制到缓冲区。函数的一般格式如下：

```
FILE* fopen(const char*, const char*);
```

其中，返回值类型 FILE* 表示该函数的返回值为文件指针类型，第 1 个参数为文件名，第 2 个参数为文件打开模式。例如，系统中打开文件的语句如下：

```
fp=fopen("D:\\demo\\Salary.dat", "ab+");
```

文件正常打开时，函数返回值指向该文件的文件指针；文件打开失败时，函数返回 NULL。因此，可以根据函数的返回值是否为空，判断文件是否正常打开。在教师工资管理系统中，函数打开文件的完整语句如下：

```
FILE *fp;
fp=fopen("D:\\demo\\Salary.dat", "ab+"); /*以追加方式打开 D:\demo\Salary.dat
                                           二进制文件*/
if(fp==NULL) /*判断文件是否打开，如果文件没打开，则退出系统*/
{
    printf("Can not open file!");
    exit(0);
}
```

在打开文件的路径中使用了两个反斜杠(\\)。因为单个反斜杠是转义字符的起始符，所以使用两个反斜杠作为路径目录、子目录和文件名主干之间的分隔符。

第 2 个参数为文件打开模式，其符号及功能如表 10-1 所示。

表 10-1 文件打开模式的符号及功能

符 号	功 能
r(只读)	打开一个 ASCII 文件，只能读取其中的数据
w(只写)	创建并打开一个 ASCII 文件，只能向其写入数据
a(追加)	打开或创建一个 ASCII 文件，在文件末尾添加数据
rb(只读)	打开一个二进制文件，只能读取其中的数据

续表

符 号	功 能
wb(只写)	创建并打开一个二进制文件，只能向其写入数据
ab(追加)	打开或创建一个二进制文件，在文件的末尾添加数据
r+(读与写)	打开一个 ASCII 文件，可读取或写入数据
w+(读与写)	创建并打开一个 ASCII 文件，可读取或写入数据
a+(读与写)	打开或创建一个 ASCII 文件，可读取或在文件末尾添加数据
rb+(读与写)	打开一个二进制文件，可读取或写入数据
wb+(读与写)	创建并打开一个二进制文件，可读取或写入数据
ab+(读与写)	打开或创建一个二进制文件，可读取或在文件末尾添加数据

在文件打开模式中，以 **r** 开头的打开模式，只能对已经存在的文件进行操作，不能创建新文件。以 **w** 开头的打开模式，打开文件时，如果文件已经存在，将覆盖已有数据；如果打开的文件不存在，则新建文件。以 **a** 开头的打开模式，要先检查文件是否存在，如果存在，则打开文件；如果不存在，则新建文件。

10.2.2 关闭文件函数

在完成一个文件的使用后，应将其关闭，防止文件被误用或数据丢失，同时及时释放内存，减少系统资源的占用。C 语言提供用于关闭文件的函数：**fclose()**。函数的一般格式如下：

```
int fclose(FILE*);
```

函数的功能是：关闭文件，将缓冲区中的数据写入磁盘，同时自动释放分配给此文件的缓冲区。该声明的返回值类型为 **int**。如果成功关闭，则返回 **0**，否则返回 **EOF(-1)**。函数中的参数表示待关闭的文件，此参数只能使用文件指针，不能使用具体的文件名。例如，系统中读取文件内容的函数为 **OpenFile()**，在数据读取完成后，将 **fp** 关闭，代码如下：

```
int OpenFile(int cnt)
{
    FILE *fp;
    int i=0;
    fp=fopen("D:\\demo\\Salary.dat","ab+"); /*以追加方式打开D:\\demo\\Salary.dat
                                           二进制文件*/
    ...                                     /*读取数据省略*/
    fclose(fp);                             /*关闭文件*/
    return cnt;                             /*返回文件中的记录数*/
}
```

关闭 **fp** 指向的文件，同时 **fp** 不再指向该文件。



10.3 文件的读与写

打开文件的目的是就要从文件读或向文件写数据。根据读、写内容形式的不同，分别定义不同的函数进行操作。**fputc()** 函数和 **fgetc()** 函数是对单个字符进行操作，**fputs()** 函数和

fgets() 函数是对字符串进行操作, fprintf() 函数和 fscanf() 函数是进行格式化操作, fread() 函数和 fwrite() 函数是对数据块进行操作。

10.3.1 文件的写函数

“文件的写”是将内存中变量保存的数据存储到缓冲区文件中。文件分为 ASCII 文件和二进制文件, 因为它们的存放形式不同, 所以写文件的方法也不一样。

1. fputc() 函数

fputc() 函数用于向文件写入一个字符, 一般格式如下:

```
int fputc(char ch, FILE *fp);
```

第 1 个参数表示写入文件的字符, 第 2 个参数表示写入数据的文件, int 表示返回值的类型。如果执行成功, 返回值是所写字符; 如果失败, 返回值是 EOF。

【案例 10-1】 使用 fputc() 函数向磁盘文件“D:\C 语言案例教程\10-1\10-1.txt”写入“Hello world!”。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char ch;
    fp=fopen("D:\\C 语言案例教程\\10-1\\10-1.txt", "w");
    ch=getchar();
    while(ch!='\n')        /*以输入回车符作为结束标志从键盘获取字符*/
    {
        fputc(ch, fp);    /*使用 fputc() 函数将 ch 字符写入 fp 指向的磁盘文件*/
        ch=getchar();
    }
    fclose(fp);
    return 0;
}
```

265

在程序运行窗口输入“Hello world!”, 如图 10-5 所示。在 D 盘“C 语言案例教程\10-1”路径下打开“10-1.txt”文件, 可以看到文件的内容为“Hello world!”, 如图 10-6 所示。

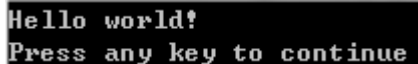


图 10-5 输入“Hello World!”

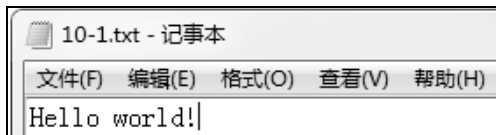


图 10-6 “10-1.txt”文件的内容

在程序运行过程中, 只要键盘输入的字符不是回车符, while 循环就将键盘输入的字符写入 fp 指向的文件。需要注意的是, 案例 10-1 中, 在 D 盘中必须有“C 语言案例教程”文件夹, 在此文件夹中必须有“10-1”文件夹, 其中必须有“10-1”文件, 否则打开失败。

2. fputs() 函数

fputs() 函数可以向文件写入一个字符串，一般格式如下：

```
int fputs(const char*, FILE *fp);
```

函数的功能是：将字符串指针指向的字符串写入文件，其中字符串的结束标志'\0'不写入。第 1 个参数是要写入文件的字符串，第 2 个参数是被写入字符串的文件，int 表示函数的返回值类型。如果执行成功，则文件位置指针会自动后移，并且返回非负整数；如果执行失败，则返回值为 EOF。

【案例 10-2】 使用 fputs() 函数向磁盘文件“D:\C 语言案例教程\10-2\10-2.txt”写入“Hello world!”。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    char str[100];
    fp=fopen("D:\\C 语言案例教程\\10-2\\10-2.txt", "w");
    gets(str);
    fputs(str, fp); /*使用 fputs() 函数将 str 字符数组中的字符串写入 fp 指向的磁盘文件*/
    fclose(fp);
    return 0;
}
```

程序执行时，先将键盘输入的字符串保存在 str 字符数组中，然后再调用 fputs() 函数将 str 字符数组中的字符串写入“D:\C 语言案例教程\10-2\10-2.txt”，执行的结果与案例 10-1 完全一致。

3. fprintf() 函数

fprintf() 是格式化输出函数，用法与 printf() 类似，但是输出对象不是终端屏幕，而是磁盘文件。一般格式如下：

```
int fprintf(FILE *fp, const char *format, ...);
```

函数的功能是：按照 format 格式字符串的格式，将输出列表项中的内容输出到文件指针指向的文件中。第 1 个参数表示文件指针，该指针指向需要写入字符串的文件，第 2 个参数表示输出格式，可以参照 printf() 函数的格式要求，返回值类型 int 表示函数返回值的类型为整型，是写入文件的字符个数。

【案例 10-3】 教师工资管理系统中有教师工资结构体变量 ts，其各成员的值如图 10-7 所示。将相关信息用 fprintf() 函数写入“D:\C 语言案例教程\10-3\10-3.txt”文件。

编号	姓名	职称	基本工资	课时数	课时费	扣款	工资
1008	李飞	副教授	3800	64	100	932	5268

图 10-7 教师工资结构体变量各成员的值

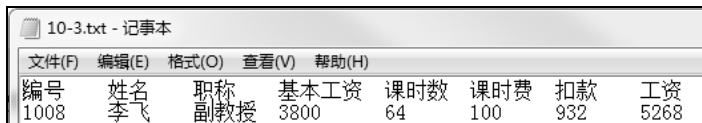
```
#include <stdio.h>
#include <string.h>
```

```

struct TeaSalary
{
    int TeaNum;           /*教师编号*/
    char TeaName[13];     /*教师姓名*/
    char TeaTitle[7];     /*教师职称*/
    int BasePay;          /*基本工资*/
    int ClassHours;       /*课时数*/
    int ClassHoursSubsidy; /*每节课课时费*/
    int Debit;            /*扣款*/
    int Salary;           /*工资*/
}ts;
int main()
{
    FILE *fp;
    fp=fopen("D:\\C 语言案例教程\\10-3\\10-3.txt","w");
    ts.TeaNum=1008;
    strcpy(ts.TeaName,"李飞"); /*将字符串赋值给字符串成员，必须使用 strcpy()函数*/
    strcpy(ts.TeaTitle,"副教授");
    ts.BasePay=3800;
    ts.ClassHours=64;
    ts.ClassHoursSubsidy=100;
    ts.Debit=932;
    ts.Salary=5268;
    fprintf(fp,"编号  姓名  职称  基本工资  课时数  课时费  扣款  工资\n");
    fprintf(fp,"%-8d%-8s%-8s%-10d%-8d%-8d%-8d\n",ts.TeaNum,ts.TeaName,
        ts.TeaTitle,ts.BasePay,ts.ClassHours,ts.ClassHoursSubsidy,
        ts.Debit,ts.Salary); /*将结构体变量各成员写入 fp 指针指向的文件*/
    fclose(fp);
    return 0;
}

```

由于程序中没有屏幕输出语句，所以程序执行完成没有任何输出，但可以在相应的文件夹中打开“10-3.txt”文件，文件的内容如图 10-8 所示。第一个 `fprintf()` 用于输出文件中第一行的标题，所以其参数中没有任何变量，第二个 `fprintf()` 用于输出结构体变量各成员的值，在格式参数后就是各成员变量。



编号	姓名	职称	基本工资	课时数	课时费	扣款	工资
1008	李飞	副教授	3800	64	100	932	5268

图 10-8 “10-3.txt”文件的内容

4. `fwrite()` 函数

对二进制文件进行写操作主要使用 `fwrite()` 函数，其写入文件的数据都是二进制数据，一般格式如下：

```
unsigned int fwrite(const void *buf, int size, int count, FILE *fp);
```

函数的功能是：从 buf 开始，分 count 次，每次 size 个字节，向 fp 对应文件的当前位置写数据。buf 是一个指针，指向将要输出数据的存储区的起始地址，size 是每次写入的字节数，count 是写入的次数，fp 是 FILE 类型的数据文件指针变量。如果执行成功，则返回 count 的值；如果执行写入的次数小于 count 次，则返回实际的次数；如果函数调用失败，则返回 0。

在教师工资管理系统中，教师工资信息写入 Salary.dat 文件就是采用此种方式，系统中保存教师工资数据的函数是 SaveFile()，程序代码如下：

```
void SaveFile(int cnt) /*保存记录*/
{
    FILE *fp;
    int i=0;
    if(l==saveflag)
    {
        fp=fopen("D:\\demo\\Salary.dat","wb");
        for(;i<cnt;i++)
            fwrite(&tea[i],sizeof(struct TeaSalary),1,fp);
        /*将结构体数组元素依次写入文件*/
        fclose(fp);
        saveflag=0; /*存盘后把存盘标志改为已存盘*/
    }
}
```

在写入数据时，第 2 个参数是每次写入的字节数，由于结构体类型 TeaSalary 中的成员数据类型在不同的编译环境中占用的内存长度不同，如果给出一个实际的数值，将降低程序的可移植性，因此在程序中使用 sizeof() 运算符计算结构体的长度。由于写入文件的是二进制数据，所以使用“记事本”打开 Salary.dat 文件会出现乱码，如图 10-9 所示。

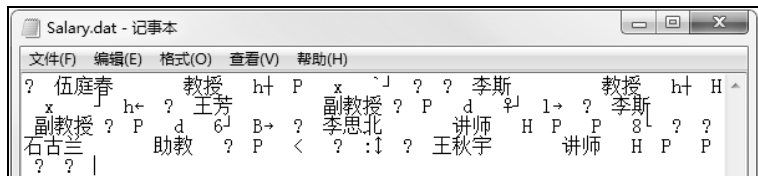


图 10-9 Salary.dat 文件的内容

在以上 4 个文件的写函数中，fputc()、fputs() 和 fprintf() 分别与 putc()、puts() 和 printf() 对应，它们的用法基本相同，都是 ASCII 文件操作函数，以字符作为操作单位，只是前者的数据流从内存流向文件，而后的数据流从内存流向屏幕；fwrite() 则是二进制文件操作函数，以二进制数据块作为操作单位。

10.3.2 文件的读函数

“文件的读”是将缓冲区中的文件内容读出，存放到内存变量中。与写文件类似，针对不同的文件形式，读文件的方式也不相同，有不同的读文件函数。

1. fgetc() 函数

函数的功能是：从文件中一次读取一个字符，一般格式如下：

```
char fgetc(FILE *fp);
```

其中, `fp` 表示读取的文件, `char` 表示返回值类型。如果执行成功, 则返回值是读取的字符; 如果执行时遇到文件结束标志, 则返回值是 `EOF`。

【案例 10-4】 使用 `fgetc()` 函数读取磁盘文件 “D:\C 语言案例教程\10-1\10-1.txt” 中的内容。

```
#include<stdio.h>
int main()
{
    FILE *fp;
    char ch;
    fp=fopen("D:\\C 语言案例教程\\10-1\\10-1.txt","r");
    ch=fgetc(fp);
    while(ch!=EOF)    /*使用 fgetc() 函数获取磁盘文件内容, 当遇到文件结束标志时停止*/
    {
        putchar(ch);    /*使用 putchar() 函数将 ch 输出到显示器*/
        ch=fgetc(fp);
    }
    putchar('\n');
    fclose(fp);
    return 0;
}
```

此案例与案例 10-1 正相反, 案例 10-1 是将键盘输入的内容存入文件, 而此案例是将先前存入文件的内容读取出来, 并输出到屏幕上, 运行结果如图 10-5 所示。当 `fgetc(fp)` 函数调用每成功执行一次, 文件指针 `fp` 就会自增 1, 指向下一个字符, 然后在 `fp` 指向文件的当前位置读取该字符, 随后再向后移动, 直到 `fgetc(fp)` 的返回值为 `EOF` 为止, 表示文件结束, 循环也结束。当然, 程序运行之前, 对应的文件 “D:\C 语言案例教程\10-1\10-1.txt” 要存在, 否则, 程序运行错误。

2. `fgets()` 函数

函数的功能是: 每次从文件中读取一行字符串, 或读取指定长度的字符串, 一般格式如下:

```
char* fgets(char *str,int lenh,FILE *fp);
```

参数 `str` 为一个字符数组, 用来存储读取的字符串; 参数 `lenh` 指定存储数据的长度, 其中包括 `lenh-1` 个字符和自动添加的结束标志 `'\0'`; 参数 `fp` 是将要读取的文件指针。如果执行成功, 则返回值为 `str` 的首地址; 如果执行失败(出错或读到文件尾), 则返回值为 `NULL`。

此函数能够从 `fp` 指向的文件中读取 `lenh-1` 个字符, 并在最后自动添加 `'\0'`, 将其放入 `str`。如果读入字符的个数不到 `lenh-1` 个就遇到文件结束标志 `EOF` 或换行符 `'\n'`, 则结束读入, 同时将换行符 `'\n'` 读入 `str`。

【案例 10-5】 使用 `fgets()` 函数读取磁盘文件 “D:\C 语言案例教程\10-2\10-2.txt” 中的内容。

```
#include<stdio.h>
#include<stdlib.h>
int main()
```

```

{
    FILE *fp;
    char str[100];
    fp=fopen("D:\\C 语言案例教程\\10-2\\10-2.txt","r");
    fgets(str,100,fp);
    puts(str);
    return 0;
}

```

此案例与案例 10-2 正相反, 案例 10-2 是将键盘输入的字符串写入文件, 而此案例是将案例 10-2 写入文件的数据读取出来, 运行结果如图 10-5 所示。程序执行时, 以读的方式打开文件, 使用 `fgets()` 函数将 `fp` 指向文件的内容传给数组 `str`, 再利用 `puts()` 函数将数组中的内容输出。

3. `fscanf()` 函数

此函数用于从文件中格式化地读取数据, 一般格式如下:

```
int fscanf(FILE *fp, const char *format, ...);
```

270

`fscanf()` 函数与 `scanf()` 函数都是输入函数, 只不过取得数据的来源不同。`scanf()` 函数从键盘获取数据, 而 `fscanf()` 函数从磁盘文件获取数据。其中, 参数 `fp` 表示读取数据的文件指针, `format` 格式字符串可参照 `scanf()` 函数的要求。`fscanf()` 函数的功能是: 从 `fp` 对应的文件的当前位置, 顺序读入一个字符序列, 按 `format` 说明的格式和类型进行转换, 并存放到对应变量单元中。如果执行成功, 则返回输入的参数个数, 否则返回 EOF。

与 `scanf()` 函数一样, 如果按照 `format` 说明的格式和类型转换后的变量是基本类型变量, 需要添加 “&”, 例如:

```
fscanf(fp, "%s,%d", work, &age);
```

由于变量 `age` 是整型变量, 所以要在其前面添加 “&”, 而 `work` 是字符数组, 其数组名就表示首地址, 所以不需要添加 “&”。

【案例 10-6】 将案例 10-3 李飞教师的工资信息从 “D:\C 语言案例教程\10-3\10-3.txt” 文件中读取出来, 存放到 `TeaSalary` 结构体类型变量 `ts` 中。

```

#include<stdio.h>
#include<string.h>
struct TeaSalary
{
    int TeaNum;           /*教师编号*/
    char TeaName[13];      /*教师姓名*/
    char TeaTitle[7];      /*教师职称*/
    int BasePay;           /*基本工资*/
    int ClassHours;       /*课时数*/
    int ClassHoursSubsidy; /*每节课课时费*/
    int Debit;             /*扣款*/
    int Salary;            /*工资*/
}ts;
int main()

```



```

{
    FILE *fp;
    char head[100];
    fp=fopen("D:\\C 语言案例教程\\10-3\\10-3.txt","r");
    fgets(head,100,fp);
    fscanf(fp,"%d%s%d%d%d%d",&ts.TeaNum,ts.TeaName,ts.TeaTitle,
            &ts.BasePay,&ts.ClassHours,&ts.ClassHoursSubsidy,&ts.Debit,
            &ts.Salary);
    printf("%s",head);
    printf("%-8d%-8s%-8s%-10d%-8d%-8d%-8d%-8d\n",ts.TeaNum,ts.TeaName,
            ts.TeaTitle,ts.BasePay,ts.ClassHours,ts.ClassHoursSubsidy,
            ts.Debit,ts.Salary);
    fclose(fp);
    return 0;
}

```

程序的运行结果如图 10-10 所示。程序将文件打开后，使用 `fgets()` 函数将第 1 行的标题以字符串的形式直接读取到 `head[]` 字符数组中，并使用 `printf()` 函数原样输出到屏幕上；再使用 `fscanf()` 函数将文件中李飞老师的工资信息读取到结构体类型变量 `ts` 的各成员中，由于其编号、基本工资、课时数等成员类型是整型，所以需要在 `ts` 变量前添加“&”，使用 `printf()` 函数将 `ts` 变量中各成员的值输出，最后再关闭文件，结束程序运行。

编号	姓名	职称	基本工资	课时数	课时费	扣款	工资
1008	李飞	副教授	3800	64	100	932	5268

Press any key to continue

图 10-10 运行结果

4. fread() 函数

对二进制文件进行读操作主要使用 `fread()` 函数，一般格式如下：

```
unsigned int fwrite(const void *buf,int size,int count,FILE *fp);
```

参数 `buf` 是指针类型，指向要读入数据存储区的起始地址，`size` 指每次读取的字节数，`count` 指读取的次数，`fp` 是 `FILE` 类型的数据文件指针变量。函数的功能是：从 `fp` 指向的文件的当前位置，每次读取 `size` 个字节，共读 `count` 次数据，存放到 `buf` 指向的内存中。如果执行成功，则返回值为 `count` 的值；如果执行读取的次数小于 `count` 次，则返回实际的次数；如果函数调用失败，则返回 0。

在教师工资管理系统中，`OpenFile()` 函数的功能是打开文件，以数据块为单位读取“Salary.dat”文件中的教师工资信息，`ShowRecord()` 函数的功能是显示教师工资信息，将两者结合起来就可以实现教师工资信息的读取与显示，如案例 10-7 所示。

【案例 10-7】 读取教师工资管理系统中“Salary.dat”的数据，并输出。

```

#include<stdio.h>
#include<string.h>
#define Header printf("编号 姓名 职称 基本工资 课时数 课时费 扣款 工资\n")
#define Print printf("%-8d%-8s%-8s%-10d%-8d%-8d%-8d%-8d\n",tea[p].TeaNum,

```

```

        tea[p].TeaName,tea[p].TeaTitle,tea[p].BasePay,tea[p].ClassHours,
        tea[p].ClassHoursSubsidy,tea[p].Debit,tea[p].Salary)
struct TeaSalary
{ /*省略结构体成员定义*/
}tea[100];
int OpenFile(int cnt)
{
    FILE *fp;
    int i=0;
    fp=fopen("D:\\demo\\Salary.dat","ab+"); /*以追加方式打开D:\demo\Salary.dat
        二进制文件*/
    while(!feof(fp)) /*循环读取文件中的记录，直到文件末尾结束*/
    {
        if(fread(&tea[i],sizeof(struct TeaSalary),1,fp)==1)
            /*判断 fread 函数读取数据是否成功，返回值为 1，则读取了一条记录*/
        {
            cnt++;
            i++;
        }
    }
    fclose(fp); /*关闭文件*/
    return cnt; /*返回文件中的记录数*/
}
void ShowRecord(int cnt) /*显示所有记录*/
{
    int i,p;
    if(cnt!=0) /*如果没有记录，则不输出*/
    {
        Header; /*输出标题行*/
        for(i=0;i<cnt;i++)
        {
            p=i;
            Print; /*输出一条记录*/
        }
    }
}
int main()
{
    int count=0;
    count=OpenFile(count);
    ShowRecord(count);
}

```

程序的运行结果如图 10-11 所示。

案例 10-7 中的两个函数直接使用的是教师工资管理系统中的函数，在 main() 函数中调用就可以实现相应的功能。在 OpenFile() 函数中使用 while(!feof(fp)) 循环语句，每次通过 fread() 函数将一位教师的工资信息从文件读取到结构体数组 tea[] 中。因为每次读取一个数据块，其



图 10-11 运行结果

长度刚好是结构体类型的长度，所以读取成功，其返回值就为 1，如果返回值非 1，则说明读取失败。在 ShowRecord() 函数中使用 for 循环将结构体数组中的每个元素输出。由于“Salary.dat”文件中的数据是使用二进制写函数 fwrite() 写入的数据，在读取时，必须使用 fread() 函数才能正确读取文件中的数据。

与文件的写函数一样，在 4 个读文件函数中，fgetc()、fgets()、fscanf() 函数与 getchar()、gets()、scanf() 函数对应。它们的用法基本相同，都是 ASCII 文件操作函数，以字符为操作单位，只是前者的数据流从文件流向内存，而后的数据流从键盘流向内存。fread() 函数则是二进制文件操作函数，以二进制数据块为操作单位。

10.4 其他相关函数

前面讲的函数都是顺序读、写一个文件，每完成一个字符的读、写，文件指针就指向下一个字符。文件读、写字符的位置是由文件指针指向的位置决定的，下面介绍几个关于文件指针位置定位的函数，使用它们可以实现随机文件的读、写。

1. fseek() 函数

一般调用形式如下：

```
fseek(文件类型指针, 位移量, 起始点);
```

“文件类型指针”是一个 FILE 类型的数据文件指针变量；“位移量”是一个长整型数据，如果为正值，表示从“起始点”开始向文件末尾方向移动的字节数，如果为负值，表示从“起始点”开始向文件开头方向移动的字节数；“起始点”指移动的起始位置，可以是数字或宏名代表，在 stdio.h 文件中定义，其含义见表 10-2。

表 10-2 含义表

值	常量符号	位置
0	SEEK_SET	文件开头
1	SEEK_CUR	当前位置
2	SEEK_END	文件末尾

函数的功能是：将位置指针指向从起始点开始，按位移量移动后的位置。
如果执行成功，函数的返回值为 0；如果执行失败，函数的返回值为非 0 值。
例如：

```
FILE *fp;  
...  
fseek(fp, 20L, 0);
```

表示将位置指针从文件开头位置向文件末尾位置移动 20 个字节。

```
fseek(fp, 20L, 1);
```

表示将位置指针从当前位置向文件末尾位置移动 20 个字节。

```
fseek(fp, -20L, 1);
```

表示将位置指针从当前位置向文件开头位置移动 20 个字节。

```
fseek(fp, -20L, 2);
```

表示将位置指针从文件末尾位置向文件开头位置移动 20 个字节。

`fseek()` 函数一般用于二进制文件，因为用于 ASCII 文件要进行字符转换，计算位置经常会发生混乱。

2. `ftell()` 函数

一般调用形式如下：

```
ftell(FILE *fp);
```

`fp` 是一个 `FILE` 类型的数据文件指针变量。

函数的功能是：获取位置指针当前指向的位置。

如果执行成功，返回值为位置指针的值；如果出错（如文件不存在），返回值为 -1。

使用 `ftell()` 函数的返回值，可确定当前位置指针指向的文件是否存在。

例如：

```
FILE *fp;  
long m;  
...  
m=ftell(fp);  
if(m==-1)  
    printf("FILE ERROR!");
```

将位置指针的值赋给变量 `m`，判断其返回值，当 `m=-1` 时，表示位置指针指向文件出错，输出提示内容。

3. `rewind()` 函数

一般调用形式如下：

```
rewind(FILE *fp);
```

`fp` 是一个 `FILE` 类型的数据文件指针变量。

函数的功能是：使位置指针指向当前文件的开头。

无返回值。

4. feof() 函数

一般调用形式如下：

```
feof(FILE *fp);
```

`fp` 是一个 `FILE` 类型的数据文件指针变量。

函数的功能是：判断文件指针是否指向文件末尾。

如果文件指针指向文件末尾，则返回值为 1；如果文件指针未指向文件末尾，则返回值为 0。

5. ferror() 函数

一般调用形式如下：

```
int ferror(FILE *fp);
```

`fp` 是一个 `FILE` 类型的数据文件指针变量。

函数的功能是：检查由 `fp` 指定的文件在读、写时是否出错。

如果文件在读、写时无错误，则返回值为 0；如果文件在读、写时有错误，则返回值为 1。

可以使用 `ferror()` 函数来判断读、写文件是否出错，例如：

```
fputc(ch,fp);
if(ferror(fp))
{
    printf("write error!\n");
}
```

当对文件 `fp` 进行写操作后，就可以使用 `ferror()` 函数检查先前写文件是否出错。如果返回值为 1，则说明有错误；如果返回值为 0，则说明无错误。

6. clearerr() 函数

一般调用形式如下：

```
void clearerr(FILE *fp);
```

`fp` 是一个 `FILE` 类型的数据文件指针变量。

函数的功能是：清除由 `fp` 指定文件的错误标志。

无返回值。

当使用 `ferror()` 函数判断读、写文件出错时，在处理完错误后，可及时清理错误标志，对出错检查代码进行改进，例如：

```
fputc(ch,out);
if(ferror(out))
{
    printf("write error!\n");
    clearerr(out);
}
```

在判断出写文件出错时，输出错误信息后，利用 `clearerr()` 函数清理先前写文件的错误标志。

10.5 本章小结

本章主要讲解 C 语言文件的相关概念,包括计算机中流的定义、文件的定义、文件缓冲区、文件指针等,同时讲解文件的相关操作,如文件的打开与关闭、文件的读与写、文件定位和文件状态检查等。通过本章的学习,读者应掌握 C 语言文件的基本知识与初级操作方式,并能够使用相关函数语句操作文件。

10.6 习题

一、单选题

1. 若要打开 A 盘上 user 子目录下名为 abc.txt 的 ASCII 文件进行读、写操作,下面符合此要求的函数调用是()。

- A. fopen("A:\user\abc.txt","r") B. fopen("A:\\user\\abc.txt","r+")
C. fopen("A:\user\abc.txt","rb") C. fopen("A:\\user\\abc.txt","w")

2. 若 fp 已正确定义并指向某个文件,当未遇到该文件结束标志时,函数 feof(fp) 的值为()。

- A. 0 B. 1 C. -1 D. 一个非 0 值

3. 当已经存在一个 file1.txt 文件,执行函数 fopen("file1.txt","r+")的功能是()。

- A. 打开 file1.txt 文件,清除原有的内容
B. 打开 file1.txt 文件,只能写入新的内容
C. 打开 file1.txt 文件,只能读取原有的内容
D. 打开 file1.txt 文件,可以读取和写入新的内容

4. fread(buf,64,2,fp)的功能是()。

- A. 从 fp 指向的文件中读出整数 64,并存放在 buf 中
B. 从 fp 指向的文件中读出整数 64 和 2,并存放在 buf 中
C. 从 fp 指向的文件中读出 64 个字节的字符,读两次,并存放在 buf 地址中
D. 从 fp 指向的文件中读出 64 个字节的字符,并存放在 buf 中

5. 以下程序的功能是()。

```
main()
{
    FILE *fp;
    char str[]="Beijing 2008";
    fp=fopen("file2","w");
    fputs(str,fp);
    fclose(fp);
}
```

- A. 在屏幕上显示 Being 2008

B. 把 Beijing 2008 存入 file2 文件

C. 从打印机打印出 Beijing 2008

D. 以上都不对

6. 以下程序是建立一个名为 myfile 的文件, 并将键盘输入的字符存入该文件, 当键盘输入结束时, 关闭该文件。程序中空格处应填入的正确内容是()。

```
main()
{
    FILE *fp;
    char c;
    char name[10];
    fp=fopen("myfile", _____);
    do
    {
        c=getchar();
        fputc(c,fp);
    }while(c!=EOF);
    fclose(fp);
}
```

A. "r"

B. "r+"

C. "w"

D. "w+"

7. 有以下定义和说明, 设文件中以二进制形式存有多位学生的数据, 且已经正确打开, 文件指针定位在文件开头, 若要从文件中读出 30 位学生的数据, 并放入 a 数组中, 以下语句正确的是()。

```
#include<stdio.h>
struct std
{
    char num[6];
    char name[8];
    float mark[4];
}a[30];
FILE *fp;
```

A. fread(a,sizeof(struct std),30,fp);

B. fread(&a[i],sizeof(struct std),1,fp);

C. fread(a+i,sizeof(struct std),1,fp);

D. fread(a,struct std,30,fp);

8. 设有以下结构体类型, 并且结构体数组 student 中的元素都已经有了值, 若要将这些元素写到 fp 指向的磁盘文件中, 以下形式不正确的是()。

```
struct st
{
    char name[8];
    int num;
    float s[4];
}student[20];
```

- A. fwrite(student,sizeof(struct st),20,fp);
- B. fwrite(student,20*sizeof(struct st),1,fp);
- C. fwrite(student,10*sizeof(struct s),10,fp);
- D. for (i=0;i<20;i++)

 fwrite (student+i,sizeof(struct st),1,fp);

9. 以下程序执行后, 文件 test.t 中的内容是()。

```
#include<stdio.h>
#include<string.h>
void fun(char *fname,char *st)
{
    FILE *myf;
    int i;
    myf=fopen(fname,"w");
    for(i=0;i<strlen(st);i++)
        fputc(st[i],myf);
    fclose(myf);
}
main()
{
    fun("test","new world");
    fun("test","hello,");
}
```

- A. hello,
- B. new worldhello,
- C. new world
- D. hello, rld

10. 检查由 fp 指定的文件在读、写时是否出错的函数是()。

- A. feof()
- B. ferror()
- C. clearerr(fp)
- D. ferror(fp)

11. 若执行 fopen() 函数时发生错误, 则函数的返回值是()。

- A. 地址值
- B. 0
- C. 1
- D. EOF

12. 若以 “a+” 方式打开一个已存在的文件, 则以下叙述正确的是()。

- A. 文件打开时, 原有文件内容不被删除, 位置指针移到文件末尾, 可执行添加和读操作
- B. 文件打开时, 原有文件内容被删除, 位置指针移到文件开头, 可执行重新写和读操作
- C. 文件打开时, 原有文件内容被删除, 只可执行写操作
- D. 以上各种说法皆不正确

13. 当顺利执行文件关闭操作时, fclose() 函数的返回值是()。

- A. -1
- B. true
- C. 0
- D. 1

14. 已知函数的调用形式为 “fread(buffer,size,count,fp);”, 其中 buffer 指的是()。

- A. 一个整型变量, 代表要读入的数据项总数
- B. 一个文件指针, 指向要读的文件

- C. 一个指针, 指向要读入数据的存放地址
D. 一个存储区, 存放要读的数据项
15. fscanf() 函数的正确调用形式是()。
A. fscanf(fp,格式字符串,输入表列)
B. fscanf(格式字符串,输出表列,fp)
C. fscanf(格式字符串,文件指针,输出表列)
D. fscanf(文件指针,格式字符串,输入表列)
16. fwrite() 函数的一般调用形式是()。
A. fwrite(buffer,count,size,fp)
B. fwrite(fp,size,count,buffer)
C. fwrite(fp,count,size,buffer)
D. fwirte(buffer,size,count,fp)
17. fgetc() 函数的作用是从指定文件读入一个字符, 该文件的打开方式必须是()。
A. 只写 B. 追加 C. 读或写 D. 答案 B 和 C 都正确
18. 若调用 fputc() 函数输出字符成功, 则其返回值是()。
A. EOF B. 1 C. 0 D. 输出的字符

二、填空题

1. 根据数据的组织形式, C 语言中将文件分为_____和_____两种类型。
2. 现要求以读、写方式打开一个 ASCII 文件 stu1, 则语句为_____。
3. 现要求将第 2 题中打开的文件关闭, 则语句为_____。
4. 若要用 fopen() 函数打开一个新的二进制文件, 该文件既能读也能写, 则语句为_____。
5. 以下程序将从终端读入的 10 个整数以二进制方式写入一个名为 bi.dat 的新文件, 请在程序的对应位置填空。

```
#include<stdio.h>
FILE *fp;
main()
{ int i,j;
  if((fp=fopen(_____, "wb"))==NULL) exit(0);
  for(i=0;i<10;i++)
  { scanf("%d",&j);
    fwrite(&j,sizeof(int),1,_____);
  }
  fclose(fp);
}
```

6. 以下程序用来统计文件中字符的个数, 请在程序的对应位置填空。

```
#include"stdio.h"
main()
{ FILE *fp; long num=0L;
  if((fp=fopen("fname.dat", "r"))==NULL)
```

```

        {printf("Open error\n");exit(0);}
        while( _____ )
        {fgetc(fp);num++;}
        printf("num=%ld\n",num-1);
        fclose(fp);
    }

```

7. 以下程序中用户由键盘输入一个文件名, 然后输入一串字符(用#结束输入)存放到生成的 ASCII 文件中, 并将字符的个数写到文件尾部, 请在程序的对应位置填空。

```

#include<stdio.h>
main()
{   FILE *fp;
    char ch,fname[32];
    int count=0;
    printf("Input the filename: "); scanf("%s",fname);
    if((fp=fopen(_____, "w+"))==NULL)
    {   printf("Can't open file: %s \n",fname); exit(0);}
    printf("Enter data: \n");
    while((ch=getchar())!="#{fputc(ch, fp);count++;}
        fprintf(_____, "\n%d\n", count);
    fclose(fp);
}

```

8. 以下程序的功能是: 删除字符串中的非数字字符, 然后输入文件 file.txt, 并保存, 请在程序的对应位置填空。

```

#include<stdio.h>
#include<stdlib.h>
void main(void)
{
    FILE *fp;
    char str[100];
    int i,j;
    if((fp=fopen("file.txt", "w"))==NULL)
    {
        printf("Can't open the file!\n");
        exit(0);
    }
    printf("Please input a string:\n");
    gets(str);
    for(i=0,j=0;str[i]!='\0';i++)
    {
        if(str[i]>='0'&&str[i]<='9')
        {
            str[j]=_____ ;
            fputc(str[j],fp);
            _____ ;
        }
    }
}

```

```

    }
    str[j]='\0';
    fputc(str[j],fp);
    puts(str);
    fclose(_____);
}

```

9. 以下程序的功能是：输出 test.dat 的长度，请在程序的对应位置填空。

```

#include<stdio.h>
int main()
{
    FILE *myf;
    long fl;
    myf=_____;
    if(myf==NULL)
    {
        printf("Cannot open file!\n");
        return 0;
    }
    fseek(myf,0,_____);
    fl=_____;
    fclose(myf);
    printf("%d\n",fl);
    return 0;
}

```

三、上机操作题

1. 编写程序，实现从键盘输入一个字符串，将其中的小写字母全部转换成大写字母，然后输出到一个磁盘文件 test 中，并保存，输入的字符串以“!”结束。
2. 编写程序，实现从键盘输入一个字符串，并写到磁盘文件 test.txt 中。
3. 编写程序，实现将已有的两个文件 test1.txt 和 test2.txt 中的数据合并，然后存放到文件 test3.txt 中。
4. 在 student.dat 中存储了 10 个整型数据，编写程序，实现将这 10 个整型数据中的偶数输出到 result.dat 中。要求：使用函数 fun() 判断一个数是否是偶数，并将该函数放在头文件 function.h 中，供主函数调用。

附录 A ASCII 码表

十进制 ASCII 码	字 符	十进制 ASCII 码	字 符	十进制 ASCII 码	字 符
0	NUL	43	+	86	V
1	SOH (^A)	44	,	87	W
2	STX (^B)	45	-	88	X
3	SX (^C)	46	.	89	Y
4	EOT (^D)	47	/	90	Z
5	EDQ (^E)	48	0	91	[
6	ACK (^F)	49	1	92	\
7	BEL (bell)	50	2	93]
8	BS (^H)	51	3	94	^
9	HT (^I)	52	4	95	-
10	LF (^J)	53	5	96	`
11	VT (^K)	54	6	97	a
12	FF (^L)	55	7	98	b
13	CR (^M)	56	8	99	c
14	SO (^N)	57	9	100	d
15	SI (^O)	58	:	101	e
16	DLE (^P)	59	;	102	f
17	DC1 (^Q)	60	<	103	g
18	DC2 (^R)	61	=	104	h
19	DC3 (^S)	62	>	105	i
20	DC4 (^T)	63	?	106	j
21	NAK (^U)	64	@	107	k
22	SYN (^V)	65	A	108	l
23	ETB (^W)	66	B	109	m
24	CAN (^X)	67	C	110	n
25	EM (^Y)	68	D	111	o
26	SUB (^Z)	69	E	112	p
27	ESC	70	F	113	q
28	FS	71	G	114	r
29	GS	72	H	115	s
30	RS	73	I	116	t
31	US	74	J	117	u
32	space (空格)	75	K	118	v
33	!	76	L	119	w
34	"	77	M	120	x
35	#	78	N	121	y
36	\$	79	O	122	z
37	%	80	P	123	{
38	&	81	Q	124	
39	'	82	R	125	}
40	(83	S	126	~
41)	84	T	127	del
42	*	85	U		

附录 B 运算符的优先级和结合性

优 先 级	运 算 符	含 义	运 算 类 型	结 合 方 向
1	() [] -> •	圆括号、函数参数表 数组元素下标 指向结构体成员 引用结构体成员		自左向右
2	! ~ ++ -- - * & (类型标识符) sizeof	逻辑非 按位取反 自增、自减 求负 间接寻址运算符 取地址运算符 强制类型转换运算符 计算字节数运算符	单目运算	自右向左
3	* / %	乘、除、整数求余	双目算术运算	自左向右
4	+ -	加、减	双目算术运算	自左向右
5	<< >>	左移、右移	位运算	自左向右
6	< <= > >=	小于、小于等于 大于、大于等于	关系运算	自左向右
7	= = ! =	等于、不等于	关系运算	自左向右
8	&	按位与	位运算	自左向右
9	^	按位异或	位运算	自左向右
10		按位或	位运算	自左向右
11	&&	逻辑与	逻辑运算	自左向右
12		逻辑或	逻辑运算	自左向右
13	?:	条件运算符	三目运算	自右向左
14	= += -= *= /= %= &= ^= = <<= >>=	赋值运算符 复合的赋值运算符	双目运算	自右向左
15	,	逗号运算符	顺序求值运算	自左向右

附录 C 常用 ANSI C 标准库函数

不同的 C 编译系统提供的标准库函数的数目、函数名及函数功能并不完全相同，以下只列出 ANSI C 标准提供的一些常用库函数。

读者在编程时若用到其他库函数，请查阅系统的库函数手册。

1. 数学函数

使用数学函数时，应该在源文件中包含头文件“math.h”。

函 数 名	函数和形参类型	功 能	返 回 值	说 明
acos	double acos(x) double x;	计算 arccos(x) 的值	计算结果	$-1 \leq x \leq 1$
asin	double asin(x) double x;	计算 arcsin(x) 的值	计算结果	$-1 \leq x \leq 1$
atan	double atan(x) double x;	计算 arctan(x) 的值	计算结果	
atan2	double atan2(x,y) double x,y;	计算 arctan(x/y) 的值	计算结果	
cos	double cos(x) double x;	计算 cos(x) 的值	计算结果	x 的单位为弧度
cosh	double cosh(x) double x;	计算 x 的双曲线余弦 cosh(x) 的值	计算结果	
exp	double exp(x) double x;	求 e^x 的值	计算结果	
fabs	double fabs(x) double x;	求 x 的绝对值	计算结果	
floor	double floor(x) double x;	求不大于 x 的最大整数	该整数的双精度实数	
fmod	double fmod(x,y) double x,y;	求整除 x/y 的余数	返回余数的双精度值	
frexp	double frexp(val,eptr) double val; int *eptr;	把一个浮点数分解为尾数和指数	返回尾数	
log	double log(x) double x;	求 $\log_e x$ ，即 $\ln x$	计算结果	$x > 0$
log10	double log10(x) double x;	求 $\log_{10} x$	计算结果	$x > 0$
modf	double modf(val,iptr) double val; double *iptr;	把双精度数 val 分解为整数部分和小数部分，把整数部分存到 iptr 指向的单元中	val 的小数部分	
sin	double sin(x) double x;	计算 sin x 的值	计算结果	x 的单位为弧度
sinh	double sinh(x) double x;	计算 x 的双曲线正弦函数 sinh(x) 的值	计算结果	
sqrt	double sqrt(x) double x;	计算非负实数的平方根	计算结果	$x \geq 0$
tanh	double tanh(x) double x;	计算 x 的双曲线正切函数 tanh(x) 的值	计算结果	

2. 字符处理函数

ANSI C 标准要求在使用字符处理函数时，应包含头文件“ctype.h”。

函 数 名	函数和形参类型	功 能	返 回 值
isalnum	int isalnum (ch) int ch;	检查 ch 是否为字母(alpha)或数字(numeric)	是字母或数字，返回 1；不是，返回 0
isalpha	int isalpha (ch) int ch;	检查 ch 是否为字母	是，返回 1；不是，返回 0
isctrl	int isctrl (ch) int ch;	检查 ch 是否为控制字符(ASCII 码在 0 和 0x1F 之间)	是，返回 1；不是，返回 0
isdigit	int isdigit (ch) int ch;	检查 ch 是否为数字(0~9)	是，返回 1；不是，返回 0
isgraph	int isgraph (ch) int ch;	检查 ch 是否为可打印字符(ASCII 码在 33~126 之间，不包括空格)	是，返回 1；不是，返回 0
islower	int islower (ch) int ch;	检查 ch 是否为小写字母(a~z)	是，返回 1；不是，返回 0
isprint	int isprint (ch) int ch;	检查 ch 是否为可打印字符(ASCII 码在 32~126 之间，不包括空格)	是，返回 1；不是，返回 0
ispunct	int ispunct (ch) int ch;	检查 ch 是否为标点字符(不包括空格)，即除字母、数字和空格以外的所有可以打印的字符	是，返回 1；不是，返回 0
isspace	int isspace (ch) int ch;	检查 ch 是否为空格、跳格符(制表符)或换行符	是，返回 1；不是，返回 0
isupper	int isupper (ch) int ch;	检查 ch 是否为大写字母(A~Z)	是，返回 1；不是，返回 0
isxdigit	int isxdigit (ch) int ch;	检查 ch 是否为一个十六进制数字(即 0~9，或 A~F，或 a~f)	是，返回 1；不是，返回 0
tolower	int tolower (ch) int ch;	将 ch 字符转换为小写字母	返回 ch 代表的字符的小写字母
toupper	int toupper (ch) int ch;	将 ch 字符转换为大写字母	与 ch 相应的大写字母

3. 字符串处理函数

ANSI C 标准要求在使用字符串处理函数时，应包含头文件“string.h”。

函 数 名	函数和形参类型	功 能	返 回 值
memcmp	int memcmp (buf1,buf2,count) const void *buf1,*buf2; unsigned int count;	比较 buf1 和 buf2 指向的数组的前 count 个字符	buf1<buf2，返回负数； buf1=buf2，返回 0； buf1>buf2，返回正数
memcpy	void *memcpy (to,from,count) void *to; const void *from; unsigned int count;	从 from 指向的数组向 to 指向的数组复制 count 个字符，如果两数组重叠，不定义该数组的行为	返回指向 to 的指针
memmove	void *memmove (to,from,count) void *to; const void *from; unsigned int count;	从 from 指向的数组向 to 指向的数组复制 count 个字符，如果两数组重叠，则复制仍进行，但把内容放入 to 后修改 from	返回指向 to 的指针
memset	void *memset (buf,ch,count) void *buf; int ch; unsigned int count;	把 ch 的低字节复制到 buf 指向的数组的前 count 个字节处，常用于把某个内存区域初始化为已知值	返回 buf 指针

续表

函 数 名	函数和形参类型	功 能	返 回 值
strcat	char *strcat(str1,str2) char *str1; const char *str2;	把字符串 str2 连接到 str1 后面, 在新形成的 str1 串后面添加一个'\0', 原 str1 后面的'\0'被覆盖。因无边界检查, 调用时应保证 str1 的空间足够大, 能存放 str1 和 str2 两个串的内容	返回 str1 指针
strcmp	int strcmp(str1,str2) const char *str1,*str2;	按字典顺序比较 str1 和 str2	str1<str2, 返回负数; str1=str2, 返回 0; str1>str2, 返回正数
strcpy	char *strcpy(str1,str2) char *str1; const char *str2;	把 str2 指向的字符串复制到 str1 中, str2 必须是结束标志为'\0'的字符串指针	返回 str1 指针
strlen	unsigned int strlen(str) const char *str;	统计字符串 str 中字符的个数(不包括结束标志)	返回字符个数
strncat	char *strncat(str1,str2,count) char *str1; const char *str2; unsigned int count;	把字符串 str2 中不多于 count 个的字符连接在 str1 后面, 并以'\0'终止该串, 原 str1 后面的'\0'被 str2 的第 1 个字符覆盖	返回 str1 指针
strncmp	int strncmp(str1,str2,count) constchar *str1,*str2; unsigned int count;	按字典顺序比较两个字符串 str1 和 str2 中的不多于 count 个的字符	str1<str2, 返回负数; str1=str2, 返回 0; str1>str2, 返回正数
strncpy	char *strncpy(str1,str2,count) char *tr1; const char *str2; unsigned int count;	把 str2 指向的字符串中的 count 个字符复制到 str1 中, str2 必须是终止标志为'\0'的字符串的指针, 如果 str2 指向的字符串少于 count 个字符, 则将'\0'加到 str1 的尾部, 直到满足 count 个字符串为止, 如果 str2 指向的字符串长度大于 count 个字符, 则串 str1 不用'\0'结尾	返回 str1 指针
strstr	char *strstr(str1,str2) char *str1,*str2;	找出 str2 字符串在 str1 字符串中第 1 次出现的位置(不包括 str2 的串结束标志)	返回该位置的指针, 若找不到, 则返回指针

4. 缓冲文件系统的输入/输出函数

使用以下缓冲文件系统的输入/输出函数时, 应该在源文件中包含头文件“stdio.h”。

函 数 名	函数和形参类型	功 能	返 回 值
clearerr	void clearerr(fp) FILE *fp;	清除文件指针错误指示器	无
fclose	int fclose(fp) FILE *fp;	关闭 fp 所指的文件, 释放文件缓冲区	成功, 返回 0, 否则返回非 0
feof	int feof(fp) FILE *fp;	检查文件是否结束	遇文件结束标志返回非 0, 否则返回 0
ferror	int ferror(fp) FILE *fp;	检查 fp 指向的文件中的错误	无错时, 返回 0; 有错时, 返回非 0
fflush	int fflush(fp) FILE *fp;	如果 fp 指向的文件是“写打开”的, 则将输出缓冲区中的内容写入文件; 若文件是“读打开”的, 则清除输入缓冲区中的内容。在这两种情况下, 文件维持打开不变	成功, 返回 0; 出现错误时, 返回 EOF

续表

函 数 名	函数和形参类型	功 能	返 回 值
fgetc	int fgetc (fp) FILE *fp;	从 fp 指定的文件中取得下一个字符	返回所得的字符, 若读入出错, 返回 EOF
fgets	char *fgets (buf,n,fp) char *buf; int n; FILE *fp;	从 fp 指向的文件读取一个长度为 n-1 的字符串, 存入起始地址为 buf 的空间	返回地址 buf, 若遇文件结束标志或出错, 返回 NULL
fopen	FILE *fopen (filename,mode) const char *filename,*mode;	以 mode 指定方式打开名为 filename 的文件	成功, 返回一个文件指针; 失败, 返回 NULL 指针, 错误代码在 errno 中
fprintf	int fprintf (fp,format,args,...) FILE *fp; const char *format;	把 args 的值以 format 指定的格式输出到 fp 指定的文件中	实际输出的字符数
fputc	int fputc (ch,fp) char ch; FILE *fp	将字符 ch 输出到 fp 指向的文件中	成功, 返回该字符, 否则 返回 EOF
fputs	int fputs (str,fp) const char *str; FILE *fp;	将 str 指向的字符串输出到 fp 指定的文件中	返回 0, 若出错, 返回非 0
fread	int fread (pt,,size,n,fp) char *pt; unsigned int size,n; FILE *fp;	从 fp 指定的文件中读取长度为 size 的 n 个数据项, 存到 pt 指向的内存区	返回所读的数据项个数, 若遇文件结束或出错, 返回 0
fscanf	int fscanf (fp,format,args,...) FILE *fp; char format;	从 fp 指定的文件中, 按 format 给定的格式, 将输入数据送到 args 指向的内存单元中 (args 是指针)	已输入的数据个数
fseek	int fseek (fp,offset,base) FILE *fp; long int offset; int base;	将 fp 指向的文件的位置指针移到以 base 指出的位置为基准, offset 为位移量的位置	返回当前位置, 否则返回 -1
ftell	long ftell (fp); FILE *fp;	返回 fp 指向的文件中的读/写位置	返回 fp 指向的文件中的读 /写位置
fwrite	unsigned int fwrite (prt,size,n,fp) const char *ptr ; unsigned int size,n; FILE *fp;	把 ptr 指向的 n*size 个字节输出到 fp 指向的文件中	写到 fp 文件中的数据项的 个数
getc	int getc (fp) FILE *fp;	从 fp 指向的文件中读入一个字符	返回所读的字符, 若文件 结束或出错, 返回 EOF
getchar	int getchar ()	从标准输入设备读取并返回下一个字符	返回所读字符, 若文件结 束或出错, 返回-1
gets	char *gets (str) char *str;	从标准输入设备读入字符串, 放到 str 指向的字符数组中, 一直读到接收新行符或 EOF 时为止, 新行符不作为读入串的内容, 变成'\0'后作为该字符串的结尾	成功, 返回 str 指针; 否则 返回 NULL 指针
perror	void perror (str) const char *str;	向标准错误输出字符串 str, 并随后附上冒号及全局变量 errno 代表的错误消息的文字说明	无
printf	int printf (format,args,...) const char *format;	将 args 的值输出到标准输出设备	输出字符的个数, 若出错, 返回负数

续表

函 数 名	函数和形参类型	功 能	返 回 值
putc	int putc (ch ,fp) int ch; FILE *fp;	把一个字符 ch 输出到 fp 所指的文件中	输出的字符 ch, 若出错, 返回 EOF
putchar	int putchar (ch) char ch	把字符 ch 输出到标准输出设备	输出的字符, 若出错, 返回 EOF
puts	int puts (str) const char *str;	把 str 指向的字符串输出到标准输出设备, 将 '\0' 转换为回车换行	返回换行符, 若失败, 返回 EOF
rename	int rename (oldname,newname) const char *oldname,*newname;	把 oldname 所指的文件名改为由 newname 所指的文件名	成功, 返回 0, 出错, 返回 1
rewind	void rewind (fp) FILE *fp;	将 fp 指示的文件中的位置指针置于文件开头位置, 并清除文件结束标志	无
scanf	int scanf(format,args,...) const char *format	从标准输入设备按 format 指向的字符串规定的格式, 输入数据给 args 指向的单元	读入并赋给 args 的数据个数, 遇文件结束, 返回 EOF, 出错, 返回 0

5. 动态内存分配函数

ANSI C 标准建议在“stdlib.h”头文件中包含有关动态内存分配函数的信息, 也有编译系统统用“malloc.h”来包含。

函 数 名	函数和形参类型	功 能	返 回 值
calloc	void (或 char) *calloc (n,size) unsigned n; unsigned size	分配 n 个数据项的内存连续空间, 每项大小为 size 字节	分配内存单元的起始地址, 如果不成功, 返回 0
free	void free (p) void (或 char) *p	释放 p 所指的内存区	无
malloc	void (或 char) *malloc (size) unsigned size	分配 size 字节的存储区	分配内存单元的起始地址, 如果内存不够, 返回 0
realloc	void (或 char) *realloc (p,size) void (或 char) *p; unsigned size;	将 f 指出的已分配内存区的大小改为 size, size 可比原来分配的空间大或小	返回指向该内存区的指针

6. 其他常用函数

函 数 名	函数和形参类型	功 能	返 回 值
atof	#include<stdlib.h> double atof (str) const char *str	把 str 指向的字符串转换成双精度浮点值, 串中必须包含合法的浮点数, 否则返回值无定义	返回转换后的双精度浮点值
atoi	#include<stdlib.h> int atoi (str) const char *str	把 str 指向的字符串转换成整型值, 串中必须包含合法的整型数, 否则返回值无定义	返回转换后的整型值
atol	#include<stdlib.h> long int atol (str) const char *str;	把 str 指向的字符串转换成长整型值, 串中必须包含合法的整型数, 否则返回值无定义	返回转换后的长整型值
exit	#include<stdlib.h> void exit (code) int code;	执行该函数时, 程序立即正常终止, 清空和关闭任何打开的文件。程序正常退出状态由 code 等于 0 或 EXIT_SUCCESS 表示, 非 0 值或 EXIT_FAILURE 表明定义实现出错	无

续表			
函 数 名	函数和形参类型	功 能	返 回 值
rand	#include<stdlib.h> int rand (void)	产生伪随机数序列	返回 0 到 RAND_MAX 之间的随机整数，RAND_MAX 至少是 32767
srand	#include<stdlib.h> void srand (seed) unsigned int seed;	为 rand() 函数生成的伪随机数序列设置起点种子值	无
time	#include<time.h> time_t time (time_t *time)	调用时可使用空指针,也可使用指向 time_t 类型变量的指针, 若使用后者, 则该变量可被赋予日历时间	返回系统的当前日历时间; 如果系统丢失时间设置, 函数返回-1

附录 D 教师工资管理系统完整代码

教师工资管理系统的完整代码如下：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define Header printf("编号    姓名    职称    基本工资    课时数    课时费    扣款\n")
#define Print printf("%-8d%-8s%-8s%-10d%-8d%-8d%-8d\n",tea[p].TeaNum,tea[p].TeaName,tea[p].TeaTitle,tea[p].BasePay,tea[p].ClassHours,tea[p].ClassHoursSubsidy,tea[p].Debit,tea[p].Salary)
struct TeaSalary
{
    int TeaNum;           /*教师编号*/
    char TeaName[13];     /*教师姓名*/
    char TeaTitle[7];     /*教师职称*/
    int BasePay;          /*基本工资*/
    int ClassHours;       /*课时数*/
    int ClassHoursSubsidy; /*每节课课时费*/
    int Debit;            /*扣款*/
    int Salary;           /*工资*/
}tea[100];
int saveflag=0;
void Menu()
{
    printf("                                教师工资管理系统                                \n");
    printf("                                主菜单                                \n");
    printf("1 添加教师信息                                2 显示教师信息 \n");
    printf("3 修改教师信息                                4 删除教师信息 \n");
    printf("5 查找教师信息                                6 教师信息排序 \n");
    printf("7 统计教师信息                                8 保存教师信息 \n");
    printf("                                0 退出系统                                \n");
    printf("                                Ver 1.0                                \n");
}
int OpenFile(int cnt)
{
    FILE *fp;
    int i=0;
    fp=fopen("D:\\demo\\Salary.dat","ab+"); /*以追加方式打开 D:\demo\Salary.dat
                                             二进制文件*/
    if(fp==NULL) /*判断文件是否打开，如果文件没打开，则退出系统*/
    {
        printf("Can not open file!");
        exit(0);
    }
    while(!feof(fp)) /*循环读取文件中的记录，直到文件末尾结束*/
```

```

    {
        if(fread(&tea[i],sizeof(struct TeaSalary),1,fp)==1)
            /*判断 fread()函数读取数据是否成功,返回值为 1,则读取了一条记录*/
        {
            cnt++;
            i++;
        }
    }
    fclose(fp);                /*关闭文件*/
    return cnt;                /*返回文件中记录数*/
}
int Locate(int num,int cnt) /*按教师编号查询教师信息*/
{
    int p=-1,i;
    for(i=0;i<cnt;i++)
    {
        if(tea[i].TeaNum==num)
        {
            p=i;                /*记录查询到的记录的下标*/
            break;
        }
    }
    return p;                /*返回查询到记录的下标,如果返回为-1,表示记录不存在*/
}
int JudgeBasePay(char title[]) /*根据教师职称确定教师的基本工资*/
{
    int basepay;
    if(strcmp(title,"教授")==0)
        basepay=4200;
    else
        if(strcmp(title,"副教授")==0)
            basepay=3800;
        else
            if(strcmp(title,"讲师")==0)
                basepay=3400;
            else
                basepay=3000;
    return basepay;                /*返回教师基本工资*/
}
int JudgeSubsidy(char title[]) /*根据教师职称确定教师的课时费*/
{
    int subsidy;
    if(strcmp(title,"教授")==0)
        subsidy=120;
    else
        if(strcmp(title,"副教授")==0)
            subsidy=100;
        else
            if(strcmp(title,"讲师")==0)
                subsidy=80;
            else
                subsidy=60;
    return subsidy;                /*返回教师课时费*/
}

```

```

}

int CalSalary(int basepay,int classhours,int subsidy,int debit)/*计算教师工资*/
{
    int salary;
    if(classhours<=40) /*如果教师课时每月在 40 学时内, 教师工资=基本工资-扣款*/
    {
        salary=basepay-debit;
    }
    else/*如果教师课时每月超过 40 学时, 教师工资=基本工资+(课时-40)*课时费-扣款*/
    {
        salary=basepay+(classhours-40)*subsidy-debit;
    }
    return salary; /*返回工资*/
}

int Add(int cnt) /*添加教师记录*/
{
    int choice;
    int i,teaNumTemp,j,recordcnt;
    recordcnt=cnt;
    printf("开始添加教师工资信息...\n");
    for(i=cnt;i<100;i++) /*输入的教师元素下标应该从当前记录数开始, 所以 i=cnt*/
    {
        printf("教师编号: ");
        scanf("%d",&teaNumTemp);
        for(j=0;j<i;j++)/*循环比较输入的编号与数组中编号是否相同*/
        {
            if(tea[j].TeaNum==teaNumTemp) /*如果编号重复, 提示重新输入*/
            {
                printf("教师编号重复, 请重新输入: ");
                scanf("%d",&teaNumTemp);
                j=-1; /*重新输入编号后, 让 j=-1, 经过 j++后, 使其为 j=0,
                    重新从数组的第 1 个元素开始比较编号是否重复*/
            }
        }
        tea[i].TeaNum=teaNumTemp;
        getchar();
        printf("教师姓名: ");
        gets(tea[i].TeaName);
        printf("教师职称: ");
        gets(tea[i].TeaTitle);
        printf("教师课时: ");
        scanf("%d",&tea[i].ClassHours);
        printf("教师扣款: ");
        scanf("%d",&tea[i].Debit);
        tea[i].BasePay=JudgeBasePay(tea[i].TeaTitle);/*计算基本工资*/
        tea[i].ClassHoursSubsidy=JudgeSubsidy(tea[i].TeaTitle);/*计算课时费*/
        tea[i].Salary=CalSalary(tea[i].BasePay,tea[i].ClassHours,
            tea[i].ClassHoursSubsidy,tea[i].Debit);/*计算工资*/
        printf("是否继续输入? 1 继续 0 退出: ");
        scanf("%d",&choice);
        recordcnt++; /*输完一条记录后让记录数增 1*/
        if(choice!=1)
    }
}

```

```

        break;
    }
    saveflag=1;      /*输入记录,记录发生变化,saveflag=1,表示变动后的数据未保存*/
    printf("记录输入完成,按任意键返回主菜单....");
    getchar();
    getchar();
    return recordcnt; /*返回记录数*/
}
void ShowRecord(int cnt) /*显示所有记录*/
{
    int i,p;
    if(cnt!=0)           /*如果没有记录,则不输出*/
    {
        Header;          /*输出标题行*/
        for(i=0;i<cnt;i++)
        {
            p=i;
            Print;         /*输出一条记录*/
        }
    }
}
void Modify(int cnt)     /*修改教师工资信息*/
{
    int tempnum,p,teaNumTemp,j;
    printf("开始修改教师信息....\n\n");
    printf("请输入要修改教师的编号: ");
    scanf("%d",&tempnum);
    p=Locate(tempnum,cnt); /*调用按编号查询教师信息函数*/
    if(-1==p)              /*返回值-1,表示要查询的教师信息不存在*/
    {
        printf("\n 要修改的教师信息不存在,按任意键返回主菜单....");
    }
    else
    {
        printf("要修改的教师信息如下: \n\n");
        Header;
        Print;             /*显示查询到的教师信息*/
        printf("请重新输入教师信息: \n");
        /*以下是重新输入教师相关信息*/
        printf("教师编号: ");
        scanf("%d",&teaNumTemp);
        for(j=0;j<cnt;j++)
        {
            if(tea[j].TeaNum==teaNumTemp && j!=p)
            {
                printf("教师编号重复,请重新输入: ");
                scanf("%d",&teaNumTemp);
                j=-1;
            }
        }
        tea[p].TeaNum=teaNumTemp;
        getchar();
        printf("教师姓名: ");
    }
}

```

```

        gets(tea[p].TeaName);
        printf("教师职称: ");
        gets(tea[p].TeaTitle);
        tea[p].BasePay=JudgeBasePay(tea[p].TeaTitle);
        printf("教师课时: ");
        scanf("%d",&tea[p].ClassHours);
        tea[p].ClassHoursSubsidy=JudgeSubsidy(tea[p].TeaTitle);
        printf("教师扣款: ");
        scanf("%d",&tea[p].Debit);
        tea[p].Salary=CalSalary(tea[p].BasePay,tea[p].ClassHours,
                                tea[p].ClassHoursSubsidy,tea[p].Debit);
        saveflag=1;          /*记录发生变化, 存盘标志改为未存盘*/
        printf("\n 教师信息修改成功, 按任意键返回主菜单....");
    }
    getchar();
    getchar();
}
int Del(int cnt)
{
    int tempnum,p,i;
    printf("开始删除教师信息....\n\n");
    printf("请输入要删除教师的编号: ");
    scanf("%d",&tempnum);
    p=Locate(tempnum,cnt); /*调用按编号查询教师信息函数*/
    if(-1==p)
    {
        printf("\n 要删除的教师信息不存在, 按任意键返回主菜单....");
    }
    else
    {
        printf("要删除的教师信息如下: \n\n");
        Header;
        Print;
        for(i=p;i<cnt-1;i++)/*从要删除的位置开始, 用后一条记录把前一条记录覆盖*/
        {
            tea[i]=tea[i+1];
        }
        cnt--;          /*记录数在原来记录数的基础上减 1*/
        saveflag=1;      /*记录发生变化, 存盘标志改为未存盘*/
        printf("\n 教师信息修改成功, 按任意键返回主菜单.....");
    }
    getchar();
    getchar();
    return cnt;          /*返回记录数*/
}
void Query(int cnt)
{
    int choice=0,tempnum,p,count=0,i;
    char tempname[13];
    printf("开始查询教师信息.....\n");
    printf("请选择查询方式(1 按编号 2 按姓名): ");
    scanf("%d",&choice);
    switch(choice)

```



```

{
case 1:
    printf("请输入要查询教师的编号: ");
    scanf("%d", &tempnum);
    p=Locate(tempnum,cnt); /*调用按编号查询教师信息函数*/
    if(-1==p)
    {
        printf("\n 查询的教师信息不存在, 按任意键返回主菜单....");
    }
    else
    {
        printf("查询的教师信息如下: \n\n");
        Header;
        Print;
        printf("\n 教师信息查询完成, 按任意键返回主菜单....");
    }
    getchar();
    break;
case 2:
    getchar();
    printf("请输入要查询教师的姓名: ");
    gets(tempname);
    for(i=0;i<cnt;i++) /*按姓名查询可能有多条记录满足, 所以不能用 Locate() 函数*/
    {
        if(strcmp(tempname,tea[i].TeaName)==0)
            /*判断元素姓名是否与要查询的姓名相同*/

        {
            if(count==0)
            {
                Header; /*如果是第 1 条要查询的记录, 则输出行标题, 否则不输出*/
            }
            p=i;
            Print;      /*输出查询到的记录*/
            count++;     /*查询到的记录数*/
        }
    }
    if(0==count)
        printf("\n 查询的教师信息不存在, 按任意键返回主菜单....");
    else
        printf("\n 教师信息查询完成, 按任意键返回主菜单....");
    break;
default:
    printf("选择错误, 按任意键返回主菜单....");
}
getchar();
}
void Sort(int cnt)
{
    int i,j,choice=0;
    struct TeaSalary temp;
    printf("开始对教师信息进行排序....\n");
    printf("请选择排序方式(1 按编号 2 按工资): ");
    scanf("%d",&choice);

```

```

        for(i=0;i<cnt-1;i++)    /*冒泡排序*/
        {
            for(j=0;j<cnt-i-1;j++)
            {
                if(1==choice)    /*按编号升序排序*/
                {
                    if(tea[j].TeaNum>tea[j+1].TeaNum)
                    {
                        temp=tea[j];tea[j]=tea[j+1];tea[j+1]=temp;
                    }
                }
                Else                /*按工资降序排序*/
                {
                    if(tea[j].Salary<tea[j+1].Salary)
                    {
                        temp=tea[j];tea[j]=tea[j+1];tea[j+1]=temp;
                    }
                }
            }
        }
        saveflag=1;
        printf("教师信息排序如下: \n");
        ShowRecord(cnt);    /*调用输出记录函数打印排序结果*/
        printf("\n 排序完成, 按任意键返回主菜单....");
        getchar();
        getchar();
    }
}

void Total(int cnt)
{
    int i,maxsalaryp=0,minsalaryp=0;
    int cntp=0,cnta=0,cntl=0,cnto=0;
    float avesalary=0;
    for(i=0;i<cnt;i++)
    {
        avesalary+=tea[i].Salary;    /*计算工资总额*/
        if(tea[i].Salary>tea[maxsalaryp].Salary)
            maxsalaryp=i;    /*记录最高工资下标*/
        if(tea[i].Salary<tea[minsalaryp].Salary)
            minsalaryp=i;    /*记录最低工资下标*/
        if(strcmp(tea[i].TeaTitle,"教授")==0)
            cntp++;    /*统计教授人数*/
        else
        {
            if(strcmp(tea[i].TeaTitle,"副教授")==0)
                cnta++;    /*统计副教授人数*/
            else
            {
                if(strcmp(tea[i].TeaTitle,"讲师")==0)
                    cntl++;    /*统计讲师人数*/
                else
                    cnto++;    /*统计其他人数*/
            }
        }
    }
    printf("全校教师工资信息如下: \n");
    ShowRecord(cnt);    /*调用输出记录函数打印所有教师信息*/
    printf("\n 全校共有教师%d 名.\n",cnt);
}

```

```

printf("其中教授%d名, 占%.2f%%, 副教授%d名, 占%.2f%%.\n", cntp, cntp*1.0/
    cnt*100, cnta, cnta*1.0/cnt*100);
printf("讲师%d名, 占%.2f%%, 其他教师%d名, 占%.2f%%.\n", cntl, cntl*1.0/
    cnt*100, cnto, cnto*1.0/cnt*100);
printf("最高工资为%d, 是%s教师.\n", tea[maxsalary].Salary,
    tea[maxsalary].TeaName);
printf("最低工资为%d, 是%s教师.\n", tea[minsalary].Salary,
    tea[minsalary].TeaName);
printf("全校教师平均工资是%.2f.\n", avesalary/cnt);
printf("教师工资信息统计完成, 按任意键返回主菜单....");
getchar();
getchar();
}
void SaveFile(int cnt)          /*保存记录*/
{
    FILE *fp;
    int i=0;
    if(l==saveflag)
    {
        fp=fopen("D:\\demo\\Salary.dat", "wb");
        for(; i<cnt; i++)
            fwrite(&tea[i], sizeof(struct TeaSalary), 1, fp);
        fclose(fp);
        saveflag=0;          /*存盘后把存盘标志改为已存盘*/
    }
}
void ExitSystem(cnt)           /*退出系统*/
{
    int choice=0;
    if(l==saveflag)            /*如果没存盘, 提示是否存盘*/
    {
        printf("教师信息有变动, 是否保存数据?(1 保存退出 2 不保存退出): ");
        scanf("%d", &choice);
        if(l==choice)
        {
            SaveFile(cnt);    /*调用存盘函数保存文件*/
            printf("文件保存成功, 按任意键退出系统....");
        }
    }
    else
    {
        printf("按任意键退出系统....");
    }
    getchar();
    getchar();
    exit(0);
}
void main()
{
    int choice=0, count=0;
    count=OpenFile(count);    /*打开文件, 获取文件中记录数*/
    while(1)

```

```
{
    system("cls");          /*调用系统清屏功能清除屏幕信息,使光标回到左上角开始位置*/
    Menu();                 /*调用菜单函数*/
    printf("请输入你的选择: ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            count=Add(count);          /*调用添加记录函数*/
            break;
        case 2:
            if(0==count)
                printf("当前没有教师信息,按任意键返回主菜单....");
            else
                printf("开始显示教师工资信息....\n\n");
                printf("所有教师信息如下: \n");
                ShowRecord(count);      /*调用显示记录函数*/
                printf("记录显示完成,按任意键返回主菜单....");
                getchar();
                getchar();
            break;
        case 3:
            Modify(count);              /*调用修改记录函数*/
            break;
        case 4:
            count=Del(count);           /*调用删除记录函数*/
            break;
        case 5:
            Query(count);               /*调用查询记录函数*/
            break;
        case 6:
            Sort(count);                /*调用排序记录函数*/
            break;
        case 7:
            Total(count);               /*调用统计记录函数*/
            break;
        case 8:
            SaveFile(count);            /*调用存盘函数*/
            printf("文件保存成功,按任意键返回主菜单....");
            getchar();
            getchar();
            break;
        case 0:
            ExitSystem(count);          /*调用退出系统函数*/
        default:
            printf("选择错误,按任意键返回主菜单....");
            getchar();
            getchar();
    }
}
```